



[Products & APIs](#)
[Developer Connection](#)
[Docs & Training](#)
[Online Support](#)
[Community Discussion](#)
[Industry News](#)
[Solutions Marketplace](#)
[Case Studies](#)

[Training Index](#)

Advanced Programming for the Java™ 2 Platform

By Calvin Austin and Monica Pawlan
November 1999

[\[CONTENTS\]](#) [\[NEXT>>\]](#)

[Printable Page](#) 

[\[DOWNLOAD\]](#)

■ Requires login

[Early Access](#) ■
[Downloads](#)

[Bug Database](#) ■
[Submit a Bug](#)
[View Database](#)

[Newsletters](#)
[Back Issues](#)
[Subscribe](#)

[Learning Centers](#)
[Articles](#)
[Bookshelf](#)
[Code Samples](#)
[New to Java](#)
[Question of the Week](#)
[Quizzes](#)
[Tech Tips](#)
[Tutorials](#)

[Forums](#)

As an experienced developer on the Java™ platform, you undoubtedly know how fast moving and comprehensive the platform is. Its many application programming interfaces (APIs) provide a wealth of functionality for all aspects of application and system-level programming. Real-world developers never use one or two APIs to solve a problem, but bring together key functionality spanning a number of APIs. Knowing which APIs you need, which parts of which APIs you need, and how the APIs work together to create the best solution can be a daunting task.

To help you navigate the Java APIs and fast-track your project development time, this book includes the design, development, test, and deployment phases for an enterprise-worthy auction application. While the example application does not cover every possible programming scenario, it explores many common situations and the discussions leave you with a solid methodology for designing and building your own solutions.

This book is for developers with more than a beginning level of understanding of writing programs in the Java programming language. The example application is written with the Java® 2 platform APIs and explained in terms of functional hows and whys, so if you need help installing the Java platform, setting up your environment, or getting your first application to work, you should first read a more introductory book such as [Essentials of the Java Programming Language: A Hands-On Guide](#) or [The Java Tutorial](#).

Technology Centers



Note: This tutorial is available as a book from [online book sellers](#). Also, send your comments and thoughts to jdcbook@sun.com

Contents

Chapter 1: [Matching Project Requirements with Technology](#)

[Project Requirements](#)
[Choosing the Software](#)

Chapter 2: [Auction House Application](#)

[A Multi-Tiered Application with Enterprise Beans](#)
[Entity and Session Beans](#)
[Examining a Container-Managed Bean](#)
[Container-Managed finder Methods](#)

Chapter 3: [Data and Transaction Management](#)

[Bean-Managed Persistence and the JDBC™ Platform](#)
[Managing Transactions](#)
[Bean-Managed finder Methods](#)

Chapter 4: [Distributed Computing](#)

[Lookup Services](#)
[Remote Method Invocation \(RMI\)](#)
[Common Object Request Broker Architecture \(CORBA\)](#)
[JDBC™ Technology](#)
[Servlets](#)

Chapter 5: [Java Native Interface \(JNI\) Technology](#)

[JNI Example](#)
[Strings and Arrays](#)

[Other Programming Issues](#)

Chapter 6: [Project Swing: Building a User Interface](#)

[Components and Data Models](#)

[Printing API](#)

[Advanced Printing](#)

Chapter 7: [Debugging Applets, Applications, and Servlets](#)

[Collecting Evidence](#)

[Running Tests and Analyzing](#)

[Servlet Debugging](#)

[AWT Event Debugging](#)

[Analyzing Stack Traces](#)

[Version Issues](#)

Chapter 8: [Performance Techniques](#)

[Improving Performance by Design](#)

[Connection Pooling](#)

[Performance Features and Tools](#)

[Performance Analysis](#)

[Caching Client/Server Applications](#)

Chapter 9: [Deploying the Auction Application](#)

[Java Archive File Format](#)

[Solaris™ Platform](#)

[Win32 Platform](#)

Chapter 10: [More Security Topics](#)

[Signed Applets](#)

[Writing a Security Manager](#)

[Appendix A: Security and Permissions](#)

[Appendix B: Classes, Methods, and Permissions](#)

[Appendix C: SecurityManager Methods](#)

[Epilogue](#)


Acknowledgements

Special thanks to experts Isaac Elias, Daniel Liu, and Mark Horwath for their contributions to the advanced examples in the book.

Reader Feedback

[Tell us what you think of this book.](#)

[\[TOP\]](#)

Printable Page 

[This page was updated: 4-Jun-2001]

[Products & APIs](#) | [Developer Connection](#) | [Docs & Training](#) | [Online Support](#)
[Community Discussion](#) | [Industry News](#) | [Solutions Marketplace](#) | [Case Studies](#)

[Glossary](#) | [Feedback](#) | [A-Z Index](#)

For more information on Java technology
and other software from Sun Microsystems, call:
(800) 786-7638
Outside the U.S. and Canada, dial your country's
[AT&T Direct Access Number](#) first.



Copyright © 1995-2001 [Sun Microsystems, Inc.](#)
All Rights Reserved. [Terms of Use](#). [Privacy Policy](#).



[Products & APIs](#)
[Developer Connection](#)
[Docs & Training](#)
[Online Support](#)
[Community Discussion](#)
[Industry News](#)
[Solutions Marketplace](#)
[Case Studies](#)

[Training Index](#)

Printable Page 

■ Requires login

Early Access ■
[Downloads](#)

Bug Database ■
[Submit a Bug](#)
[View Database](#)

Newsletters
[Back Issues](#)
[Subscribe](#)

Learning Centers
[Articles](#)
[Bookshelf](#)
[Code Samples](#)
[New to Java](#)
[Question of the Week](#)
[Quizzes](#)
[Tech Tips](#)
[Tutorials](#)

Forums

Writing Advanced Applications

Chapter 1: Matching Project Requirements with Technology

[<<[BACK](#)] [[CONTENTS](#)] [[NEXT](#)>>]

One challenge in writing a book on advanced application development for the Java™ platform is to find a project small enough to write about, while at the same time, complex enough to warrant advanced programming techniques.

The project presented in this book is a web-based auction house. The application is initially written for the Enterprise JavaBeans™ platform. Later chapters expand the core example described here by adding advanced functionality, improvements, and alternative solutions to do some of the things you get for free when you use the Enterprise JavaBeans platform.

To keep the discussion simple, the example application has only a basic set of transactions for posting and bidding on auction items. However, the application scales to handle multiple users, provides a three-tiered transaction-based environment, controls security, and integrates legacy-based systems. This chapter covers how to determine project requirements and model the application—important steps that should always come before coding begins.

[Project Requirements and Modeling](#)
[Choosing the Software](#)

In a Rush?


This table links you directly to specific topics.

Topic	Section
Auction Demonstration	Duke's Auction

Technology Centers

Project Requirements	Interview User Base Model the Project
Modeling	House Identifies Buyers and Sellers House Determines Highest Bidder House Notifies Buyers and Sellers Anyone Searches for an Item Anyone Views Items for Sale Anyone Views Item Details Seller Posts Items for Sale Buyer Bids on Items Activity Diagram
Choosing Software	Java™ APIs

[\[TOP\]](#)

Printable Page 

[This page was updated: 4-Jun-2001]

[Products & APIs](#) | [Developer Connection](#) | [Docs & Training](#) | [Online Support](#)
[Community Discussion](#) | [Industry News](#) | [Solutions Marketplace](#) | [Case Studies](#)

[Glossary](#) | [Feedback](#) | [A-Z Index](#)

For more information on Java technology
and other software from Sun Microsystems, call:
(800) 786-7638
Outside the U.S. and Canada, dial your country's
[AT&T Direct Access Number](#) first.



Copyright © 1995-2001 [Sun Microsystems, Inc.](#)
All Rights Reserved. [Terms of Use](#). [Privacy Policy](#).



Products & APIs
 Developer Connection
 Docs & Training
 Online Support
 Community Discussion
 Industry News
 Solutions Marketplace
 Case Studies

[Training Index](#)

Printable Page 

■ Requires login

Early Access ■
 Downloads

Bug Database ■
 Submit a Bug
 View Database

Newsletters
 Back Issues
 Subscribe

Learning Centers
 Articles
 Bookshelf
 Code Samples
 New to Java
 Question of the Week
 Quizzes
 Tech Tips
 Tutorials

Forums

Writing Advanced Applications

Chapter 1 Continued: Project Requirements and Modeling

[<<BACK] [CONTENTS] [NEXT>>]

The first step in determining project requirements is to interview the user base to find out what they want in an online auction. This is an important step, and one that cannot be overrated because a solid base of user-oriented information helps you define your key application capabilities.

Chapter 2 walks through the application code, explains how the Enterprise JavaBeans platform works, and tells you how to run a live demonstration. If you have never seen or used an online auction, here are mockups of the [example auction application](#) HTML pages.

[Interview User Base](#)
[Model the Project](#)

Interview User Base

For the sake of discussion and to keep things simple, this discussion assumes interviews with the user base found auction house and user requirements, as follows:

Auction House Requirements

- Require buyer and seller information
- Bill sellers for posting items
- Record and report the day's transactions

User Requirements

- Bid on or sell an item
- Search or view items for sale
- Notify buyer and seller of sale

Model the Project

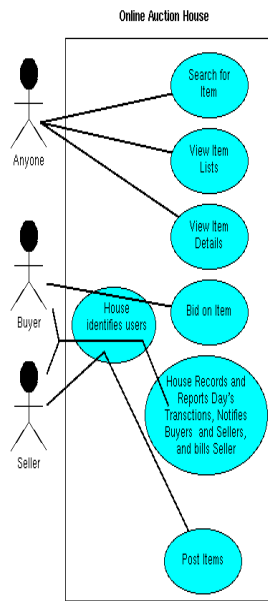
Technology Centers After analyzing the requirements, you can build a use case diagram for the application to gain a better understanding of the elements needed in the application and how they interact.

A use case diagram shows the relationships among actors and use cases within the system. A use case is a unique function in a system, and an actor is the person or software that performs the action or use case. For example, a buyer is the actor that performs the function (use case) of bidding on an auction item, and the seller is the actor that performs the use case of posting an item for auction.

Not all actors are people, though. For example, the software is the actor that determines when an item has closed, finds the highest bidder, and notifies the buyer and seller of the sale.

The [Unified Modeling Language \(UML\)](#) is the tool of choice for creating use case diagrams. The Use Case diagram below uses UML to describe the buyer and seller use cases for the online auction application.

In UML, systems are grouped into squares, actors are represented by stick figures, use cases are denoted by ovals, and the lines show how actors use the system.



The following descriptions further define the project. These descriptions are not part of UML, but are a helpful tool in project definition.

House Identifies Buyers and Sellers

An auction application is used by buyers and sellers. A buyer needs to know who the seller is to pay him or her, and the seller needs to know who the buyers are to answer product questions and to finalize the sale. So, to post or bid on an auction item, buyers and sellers are required to register. Registration needs to get the following information from buyers and sellers:

- User ID and password for buying and selling.**
- Email address so highest bidder and seller can communicate when item closes.**

Credit card information so auction can charge sellers for listing their items.

Once registered, a user can post or bid on an item for sale.

House Determines Highest Bidder

Nightly, the auction application queries the database to record and report the day's transactions. The application find items that have closed and determines the highest bidder.

House Notifies Buyers and Sellers

The auction application uses email to notify the highest bidder and seller of the sale, and debit the seller's account.

Anyone Searches for an Item

Sellers and buyers enter a search string to locate all auction items in the database.

Anyone Views Items for Sale

To popularize the auction and encourage new buyers and sellers, the application allows anyone to view auction items without requiring user ID and password identification. To keep things simple, the auction lets anyone view summarized lists of items in the following three ways:

**All items up for auction
New items listed today
Items due to close today**

Anyone Views Item Details

The summarized lists link to the following detailed information on each item. Detail information on auction items is available to anyone without identification.

**Item Summary
Auction Item number
Current price
Number of bids
Date posted for auction
Date item closes
Seller ID
Highest bid
Item description**

Seller Posts Items for Sale

To post an item for sale, a seller needs to identify himself or herself and describe the item for sale, as follows:

- User ID and password for seller identification**
- Summary description of item**
- Starting Price for bidding**
- Detailed description of item**
- Number of days item is available for bidding**

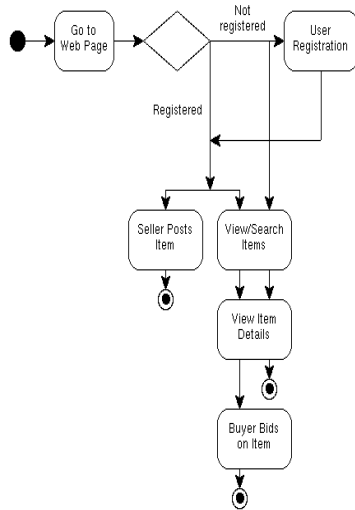
Buyer Bids on Items

The detailed summary page for each item lets registered users identify themselves and bid on the item by providing the following information:


- User ID**
- Password**
- Bid amount**

Activity Diagram

The activity diagram shows the flow of tasks within the auction house as a whole. This diagram shows the auction application. The solid black circle on the left shows the beginning of activities, and the white circles with black dots in the center denote where activities end.



[\[TOP\]](#)

Printable Page 

[This page was updated: 4-Jun-2001]

[Products & APIs](#) | [Developer Connection](#) | [Docs & Training](#) | [Online Support](#)
[Community Discussion](#) | [Industry News](#) | [Solutions Marketplace](#) | [Case Studies](#)

[Glossary](#) | [Feedback](#) | [A-Z Index](#)

For more information on Java technology
and other software from Sun Microsystems, call:
(800) 786-7638
Outside the U.S. and Canada, dial your country's
[AT&T Direct Access Number](#) first.



Copyright © 1995-2001 [Sun Microsystems, Inc.](#)
All Rights Reserved. [Terms of Use](#). [Privacy Policy](#).



[Products & APIs](#)
[Developer Connection](#)
[Docs & Training](#)
[Online Support](#)
[Community Discussion](#)
[Industry News](#)
[Solutions Marketplace](#)
[Case Studies](#)

[Printable Page](#) 

■ [Requires login](#)

[Early Access](#) ■
[Downloads](#)

[Bug Database](#) ■
[Submit a Bug](#)
[View Database](#)

[Newsletters](#)
[Back Issues](#)
[Subscribe](#)

[Learning Centers](#)
[Articles](#)
[Bookshelf](#)
[Code Samples](#)
[New to Java](#)
[Question of the Week](#)
[Quizzes](#)
[Tech Tips](#)
[Tutorials](#)

[Forums](#)

[Training Index](#)

Writing Advanced Applications

Chapter 1 Continued: Choosing the Software

[\[<<BACK\]](#) [\[CONTENTS\]](#) [\[NEXT>>\]](#)

With the application modeled and the project requirements defined, it is time to think about which Java™ APIs to use. The application is clearly client and server based because you will want to accommodate 1 to n buyers, sellers, and viewers at any one time. Because registration and auction item data must be stored and retrieved from somewhere, you will need an API for database access.

Java™ APIs

The core application can be created in a number of ways using any of the following APIs:


1. Sockets, multithreading, and JDBC™ APIs.
2. Remote Method Invocation (RMI) and JDBC APIs.
3. Enterprise JavaBeans™ platform.

Enterprise JavaBeans provides an easy way to create thin-client multitiered applications because it handles transaction and state management, multithreading, resource pooling, and other complex low-level details. The simplest way to code the auction application is with the Enterprise JavaBeans platform.

Chapter 2 explains the core application code and how to set up and run the example. With the application modeled and the project requirements defined, it is time to think about which Java™ APIs to use. The application is clearly client and server based because you will want to accommodate 1 to n buyers, sellers, and viewers at any one time. Because registration and auction item data must be stored and retrieved from somewhere, you will need an API for database access.

[\[TOP\]](#)

Technology Centers

Printable Page 

[This page was updated: 4-Jun-2001]

[Products & APIs](#) | [Developer Connection](#) | [Docs & Training](#) | [Online Support](#)
[Community Discussion](#) | [Industry News](#) | [Solutions Marketplace](#) | [Case Studies](#)

[Glossary](#) | [Feedback](#) | [A-Z Index](#)

For more information on Java technology
and other software from Sun Microsystems, call:
(800) 786-7638
Outside the U.S. and Canada, dial your country's
[AT&T Direct Access Number](#) first.



Copyright © 1995-2001 [Sun Microsystems, Inc.](#)
All Rights Reserved. [Terms of Use.](#) [Privacy Policy.](#)



- [Products & APIs](#)
- [Developer Connection](#)
- [Docs & Training](#)
- [Online Support](#)
- [Community Discussion](#)
- [Industry News](#)
- [Solutions Marketplace](#)
- [Case Studies](#)

[Training Index](#)

Writing Advanced Applications

Chapter 2: Auction House Application

[<<BACK](#) | [CONTENTS](#) | [NEXT>>](#)

[Printable Page](#) 

■ Requires login

[Early Access](#) ■
[Downloads](#)

[Bug Database](#) ■
[Submit a Bug](#)
[View Database](#)

[Newsletters](#)
[Back Issues](#)
[Subscribe](#)

[Learning Centers](#)
[Articles](#)
[Bookshelf](#)
[Code Samples](#)
[New to Java](#)
[Question of the Week](#)
[Quizzes](#)
[Tech Tips](#)
[Tutorials](#)

[Forums](#)

The example application is a web-based auction house written for the Enterprise JavaBeans™ platform. The user interface is a set of HTML pages that get input from and show information to the user. Behind the HTML pages is a servlet that passes data between the browser and the Enterprise JavaBeans server. The Enterprise JavaBeans server handles reading from and writing to the database.



This chapter describes the application code, how it works with the Enterprise JavaBeans server, and where to get a Enterprise JavaBeans server to run the example. Or, if you prefer, here is an [example mockup](#) for the auction application.

- [A Multi-Tiered Application with Enterprise Beans](#)
- [Entity and Session Beans](#)
- [Examining a Container-Managed Bean](#)
- [Container-Managed finder Methods](#)

In a Rush?

This table links you directly to specific topics.

Topic	Section

Technology Centers

A Multi-Tiered Applications with Enterprise Beans	Enterprise Beans Defined Entity and Session Beans Auction House Workings Developing and Running Applications How Multitiered Applications Work
Entity and Session Beans	Auction Servlet Entity Beans Session Beans Container Classes
Examining a Container-Managed Bean	Member Variables Create Method Entity Context Methods Load Method Store Method Connection Pooling Deployment Descriptor
Container-Managed finder Methods	AuctionServlet.searchItems BidderBean.getMatchingItemsList AuctionItemHome.findAllMatchingItems AuctionItemBean Deployment Descriptor

[\[TOP\]](#)Printable Page 

[This page was updated: 4-Jun-2001]

[Products & APIs](#) | [Developer Connection](#) | [Docs & Training](#) | [Online Support](#)
[Community Discussion](#) | [Industry News](#) | [Solutions Marketplace](#) | [Case Studies](#)

[Glossary](#) | [Feedback](#) | [A-Z Index](#)

For more information on Java technology and other software from Sun Microsystems, call: (800) 786-7638
 Outside the U.S. and Canada, dial your country's [AT&T Direct Access Number](#) first.



Copyright © 1995-2001 [Sun Microsystems, Inc.](#)
 All Rights Reserved. [Terms of Use](#). [Privacy Policy](#).

Duke's Auction



Need to clean out that old office, garage, or closet? or looking for something so unique you cannot find it anywhere—or at least not at a price you are willing to pay?

Look no further. At Duke's Auction you can post items for sale and bid what you want to pay for the items you want.

Registration

To bid on or list an item for auction, you must [register](#) first. Registration gives buyers a way to pay you and us a way to contact buyers and sellers. You only need register once, and registration is not required to browse items on the auction floor.

Auction Floor

The auction floor is open to anyone for browsing, but to bid on an item, you must be registered.

- [New auction items today](#)
- [Items closing today](#)
- [All items \(current and closed\)](#)
- [Search for Items](#)

Post Items for Auction

Once you register, you can [post items for sale](#) at auction any time you want.

[Register](#) | [New Items](#) | [Closing Items](#) | [All Items](#) | [Sell Items](#)



[Products & APIs](#)
[Developer Connection](#)
[Docs & Training](#)
[Online Support](#)
[Community Discussion](#)
[Industry News](#)
[Solutions Marketplace](#)
[Case Studies](#)

[Training Index](#)

[Printable Page](#) 

■ Requires login

[Early Access](#) ■
[Downloads](#)

[Bug Database](#) ■
[Submit a Bug](#)
[View Database](#)

[Newsletters](#)
[Back Issues](#)
[Subscribe](#)

[Learning Centers](#)
[Articles](#)
[Bookshelf](#)
[Code Samples](#)
[New to Java](#)
[Question of the Week](#)
[Quizzes](#)
[Tech Tips](#)
[Tutorials](#)

[Forums](#)

Writing Advanced Applications

Chapter 2 Continued: A Multi-Tiered Application with Enterprise Beans

[\[<<BACK\]](#) [\[CONTENTS\]](#) [\[NEXT>>\]](#)

The proliferation of internet- and intranet-based applications has created a great need for distributed transactional applications that leverage the speed, security, and reliability of server-side technology. One way to meet this need is to use a multitiered model where a thin-client application invokes business logic that executes on the server.

Normally, thin-client multitiered applications are hard to write because they involve many lines of intricate code to handle transaction and state management, multithreading, resource pooling, and other complex low-level details. And to compound the difficulties, you have to rework this code every time you write an application because the code is so low-level it is not reusable.

If you could use someone's prebuilt and pretested transaction management code or even reuse some of your own code, you would save a lot of time and energy that you could better spend solving the business problem. Well, Enterprise JavaBeans™ technology can give you the help you need. The Enterprise JavaBeans technology makes distributed transactional applications easy to write because it separates the low-level details from the business logic. You concentrate on creating the best business solution and leave the rest to the underlying architecture.

This chapter describes how to create the example auction application using the services provided by the Enterprise JavaBeans platform. Later chapters will show how you can customize these services and integrate these features into existing non-EJB applications.

[Enterprise Beans Defined](#)
[Thin-Client Programs](#)

Enterprise Beans Defined

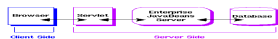
An Enterprise Bean is a simple class that provides two types of methods: business logic and lifecycle. A client program calls the business logic methods to interact with the data held on the server. The container calls the lifecycle methods to manage the Bean on the server. In addition to these two types of methods, an Enterprise Bean has an associated configuration file, called a deployment descriptor, that is used to configure the Bean at deployment time.

As well as being responsible for creating and deleting Beans the Enterprise JavaBeans server also manages transactions, concurrency, security and data persistence. Even the connections between the client and server are provided by using the RMI and JNDI APIs and servers can optionally provide scalability through thread management and caching.

The auction house example implements a complete Enterprise JavaBeans solution by providing only the business logic and using the underlying services provided by the architecture. However, you may find that the container managed services, although providing maximum portability, do not meet all your application requirements. The next chapters will show how you can provide these services in your Bean instead and also use these services in non-Enterprise Bean applications.

Thin-Client Programs

A thin client is a client program that invokes business logic running on the server. It is called *thin* because most of the processing happens on the server. In the figure below, the servlet is the thin client. It invokes Enterprise Beans that run on the Enterprise JavaBeans server. It also executes logic that creates web pages that appear in the browser.



Multitiered Architecture



Multitier architecture or three-tier architecture extends the standard two-tier client and server model by placing a multithreaded application server between the client and the database.

Client programs communicate with the database through the application server using high-level and platform independent calls. The application server responds to the client requests, makes database calls as needed into the underlying database, and replies to the client program as appropriate.

The three tiers in the web-based auction house example consists of the thin-client servlet, the Enterprise JavaBeans server (the application server), and the database server as shown in the figure.

Entity and Session Beans

There are two types of Enterprise Beans: entity Beans and session Beans. An Enterprise Bean that implements a business entity is an *entity Bean*, and an Enterprise Bean that implements a business task is a *session Bean*.

Typically, an entity Bean represents one row of persistent data stored in a database table. In the auction house example, `RegistrationBean` is an entity Bean that represents data for one registered user, and `AuctionItemBean` is an entity Bean that represents the data for one auction item. Entity Beans are transactional and long-lived. As long as the data remains, the entity Bean can access and update that data. This does not mean you need a Bean running for every table row. Instead, Enterprise Beans are loaded and saved as needed.

A session Bean might execute database reads and writes, but it is

not required. A session Bean might invoke the JDBC calls itself or it might use an entity Bean to make the call, in which case the session Bean is a client to the entity Bean. A session Bean's fields contain the state of the conversation and are transient. If the server or client crashes, the session Bean is gone. A session Bean is often used with one or more entity Beans and for complex operations on the data.

Session Beans	Entity Beans
Fields contain conversation state.	Represents data in a database.
Handles database access for client.	Shares access for multiple users.
Life of client is life of Bean.	Persists as long as data exists.
Can be transaction aware.	Transactional.
Does not survive server crashes.	Survives server crashes.
Not fine-grained data handling	Fine-grained data handling

Note: In the Enterprise Java Beans specification, Enterprise JavaBeans Server support for session Beans is mandatory. Enterprise JavaBeans server support for entity Beans was optional, but is mandatory for version 2.0 of the specification.

Auction House Workings

The diagram shows the Enterprise Beans for the auction house application and their relationship to the Enterprise JavaBeans server. The thin-client server invokes business logic in the four Enterprise Beans through their home and remote interfaces. The Enterprise JavaBeans server in this example handles the low-level details including database read and write operations.

The four Enterprise Beans in the example are:

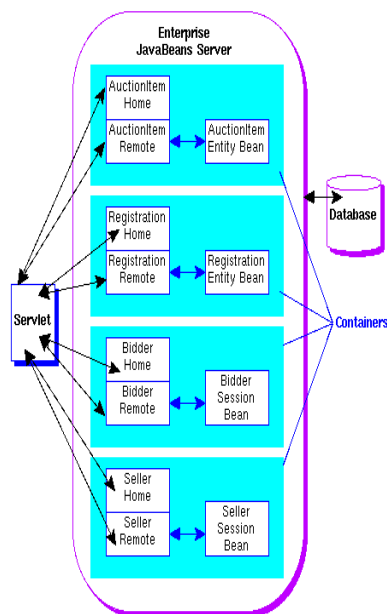
`AuctionItemBean` is an entity Bean that maintains information for an auction item.

`RegistrationBean` is an entity Bean that stores user registration information.

`BidderBean` is a session Bean that uses `AuctionItemBean` to retrieve a list of all auction items, only new items, items due

to close, and items whose summary matches a search string from the database. It also checks the user ID and password when someone places a bid, and stores new bids in the database.

`SellerBean` is a session Bean that uses `RegistrationBean` to check the user ID and password when someone posts an auction item, and `AuctionItemBean` to add new auction items to the database.



As depicted in the figure above, an entity or session Bean is really a collection of interfaces and classes. All entity and session Beans consist of a remote interface, home interface, and the Bean class. The servlet looks up the Beans's home interface running in the Enterprise JavaBeans server, uses the home interface to create the remote interface, and invokes Bean methods through the remote

interface.

An Enterprise Bean's remote interface describes the Bean's methods, or what the Bean does. A client program or another Enterprise Bean calls the methods defined in the remote interface to invoke the business logic implemented by the Bean.

An Enterprise Bean's home interface describes how a client program or another Enterprise Bean creates, finds (entity Beans only), and removes that Enterprise Bean from its container.

The container, shown in light blue (cyan), provides the interface between the Enterprise Bean and the low-level platform-specific functionality that supports the Enterprise Bean.

Developing and Running Applications

Deployment tools and an Enterprise JavaBeans server are essential to running Enterprise JavaBeans applications. Deployment tools generate containers, which are classes that provide an interface to the low-level implementations in a given Enterprise JavaBeans server. The server provider can include containers and deployment tools for their server and will typically publish their low-level interfaces so other vendors can develop containers and deployment tools for their server.

The auction house example uses the Enterprise JavaBeans server and deployment tools created by [BEA Weblogic](#).

Because everything is written to specification, all Enterprise Beans are interchangeable with containers, deployment tools, and servers created by other vendors. In fact, you might or might not write your own Enterprise Beans because it is possible, and sometimes desirable, to use Enterprise Beans written by one or more providers that you assemble into an Enterprise JavaBeans application.

How Multitiered Applications Work

The goal in a multitiered application is that the client be able to work on application data without knowing at build time where the data is stored. To make this level of transparency possible, the underlying services in a multitiered architecture use lookup

services to locate remote server objects (the Bean's remote interface object), and data communication services to move data from the client, through the remote server object, to its final destination in a storage medium.

Lookup Service

To find remote server objects at runtime, the client program needs a way to look them up. One way to look remote server objects up at runtime is to use the Java Naming and Directory Interface™ (JNDI) API. JNDI is a common interface to existing naming and directory interfaces. The Enterprise JavaBeans containers use JNDI as an interface to the Remote Method Invocation (RMI) naming service.

At deployment time, the JNDI service registers (binds) the remote interface with a name. As long as the client program uses the same naming service and asks for the remote interface by its registered name, it will be able to find it. The client program calls the `lookup` method on a `javax.naming.Context` object to ask for the remote interface by its registered name. The `javax.naming.Context` object is where the bindings are stored and is a different object from the Enterprise JavaBeans context, which is covered later.

Data Communication

Once the client program gets a reference to a remote server object, it makes calls on the remote server object's methods. Because the client program has a reference to the remote server object, a technique called data marshalling is used to make it appear as if the remote server object is local to the client program.

Data marshalling is where methods called on the remote server object are wrapped with their data and sent to the remote server object. The remote server object unwraps (unmarshalls) the methods and data, and calls the Enterprise Bean. The results of the call to the Enterprise Bean are wrapped again, passed back to the client through the remote server object, and unmarshalled.

The Enterprise JavaBeans containers use RMI services to marshal data. When the Bean is compiled, `stub` and `skeleton` files are created. The `stub` file provides the data wrapping and unwrapping configuration on the client, and the `skeleton` provides the same information for the server.

The data is passed between the client program and the server using serialization. Serialization is a way to representat Java™

objects as bytes that can be sent over the network as a stream and reconstructed on the other side in the same state they were in went originally sent.

[\[TOP\]](#)

Printable Page 

[This page was updated: 4-Jun-2001]

[Products & APIs](#) | [Developer Connection](#) | [Docs & Training](#) | [Online Support](#)
[Community Discussion](#) | [Industry News](#) | [Solutions Marketplace](#) | [Case Studies](#)

[Glossary](#) | [Feedback](#) | [A-Z Index](#)

For more information on Java technology
and other software from Sun Microsystems, call:
(800) 786-7638
Outside the U.S. and Canada, dial your country's
[AT&T Direct Access Number](#) first.



Copyright © 1995-2001 [Sun Microsystems, Inc.](#)
All Rights Reserved. [Terms of Use](#). [Privacy Policy](#).



[Products & APIs](#)
[Developer Connection](#)
[Docs & Training](#)
[Online Support](#)
[Community Discussion](#)
[Industry News](#)
[Solutions Marketplace](#)
[Case Studies](#)

[Training Index](#)

Advanced Programming for the Java™ 2 Platform

By Calvin Austin and Monica Pawlan
November 1999

[\[CONTENTS\]](#) [\[NEXT>>\]](#)

[Printable Page](#) 

[\[DOWNLOAD\]](#)

■ Requires login

[Early Access](#) ■
[Downloads](#)

[Bug Database](#) ■
[Submit a Bug](#)
[View Database](#)

[Newsletters](#)
[Back Issues](#)
[Subscribe](#)

[Learning Centers](#)
[Articles](#)
[Bookshelf](#)
[Code Samples](#)
[New to Java](#)
[Question of the Week](#)
[Quizzes](#)
[Tech Tips](#)
[Tutorials](#)

[Forums](#)

As an experienced developer on the Java™ platform, you undoubtedly know how fast moving and comprehensive the platform is. Its many application programming interfaces (APIs) provide a wealth of functionality for all aspects of application and system-level programming. Real-world developers never use one or two APIs to solve a problem, but bring together key functionality spanning a number of APIs. Knowing which APIs you need, which parts of which APIs you need, and how the APIs work together to create the best solution can be a daunting task.

To help you navigate the Java APIs and fast-track your project development time, this book includes the design, development, test, and deployment phases for an enterprise-worthy auction application. While the example application does not cover every possible programming scenario, it explores many common situations and the discussions leave you with a solid methodology for designing and building your own solutions.

This book is for developers with more than a beginning level of understanding of writing programs in the Java programming language. The example application is written with the Java® 2 platform APIs and explained in terms of functional hows and whys, so if you need help installing the Java platform, setting up your environment, or getting your first application to work, you should first read a more introductory book such as [Essentials of the Java Programming Language: A Hands-On Guide](#) or [The Java Tutorial](#).

Technology Centers



Note: This tutorial is available as a book from [online book sellers](#). Also, send your comments and thoughts to jdcbook@sun.com

Contents

Chapter 1: [Matching Project Requirements with Technology](#)

[Project Requirements](#)
[Choosing the Software](#)

Chapter 2: [Auction House Application](#)

[A Multi-Tiered Application with Enterprise Beans](#)
[Entity and Session Beans](#)
[Examining a Container-Managed Bean](#)
[Container-Managed finder Methods](#)

Chapter 3: [Data and Transaction Management](#)

[Bean-Managed Persistence and the JDBC™ Platform](#)
[Managing Transactions](#)
[Bean-Managed finder Methods](#)

Chapter 4: [Distributed Computing](#)

[Lookup Services](#)
[Remote Method Invocation \(RMI\)](#)
[Common Object Request Broker Architecture \(CORBA\)](#)
[JDBC™ Technology](#)
[Servlets](#)

Chapter 5: [Java Native Interface \(JNI\) Technology](#)

[JNI Example](#)
[Strings and Arrays](#)

[Other Programming Issues](#)

Chapter 6: [Project Swing: Building a User Interface](#)

[Components and Data Models](#)

[Printing API](#)

[Advanced Printing](#)

Chapter 7: [Debugging Applets, Applications, and Servlets](#)

[Collecting Evidence](#)

[Running Tests and Analyzing](#)

[Servlet Debugging](#)

[AWT Event Debugging](#)

[Analyzing Stack Traces](#)

[Version Issues](#)

Chapter 8: [Performance Techniques](#)

[Improving Performance by Design](#)

[Connection Pooling](#)

[Performance Features and Tools](#)

[Performance Analysis](#)

[Caching Client/Server Applications](#)

Chapter 9: [Deploying the Auction Application](#)

[Java Archive File Format](#)

[Solaris™ Platform](#)

[Win32 Platform](#)

Chapter 10: [More Security Topics](#)

[Signed Applets](#)

[Writing a Security Manager](#)

[Appendix A: Security and Permissions](#)

[Appendix B: Classes, Methods, and Permissions](#)

[Appendix C: SecurityManager Methods](#)

[Epilogue](#)


Acknowledgements

Special thanks to experts Isaac Elias, Daniel Liu, and Mark Horwath for their contributions to the advanced examples in the book.

Reader Feedback

[Tell us what you think of this book.](#)

[\[TOP\]](#)

Printable Page 

[This page was updated: 4-Jun-2001]

[Products & APIs](#) | [Developer Connection](#) | [Docs & Training](#) | [Online Support](#)
[Community Discussion](#) | [Industry News](#) | [Solutions Marketplace](#) | [Case Studies](#)

[Glossary](#) | [Feedback](#) | [A-Z Index](#)

For more information on Java technology
and other software from Sun Microsystems, call:
(800) 786-7638
Outside the U.S. and Canada, dial your country's
[AT&T Direct Access Number](#) first.



Copyright © 1995-2001 [Sun Microsystems, Inc.](#)
All Rights Reserved. [Terms of Use](#). [Privacy Policy](#).



Products & APIs
 Developer Connection
 Docs & Training
 Online Support
 Community Discussion
 Industry News
 Solutions Marketplace
 Case Studies

[Training Index](#)

Writing Advanced Applications

Chapter 2 Continued: Entity and Session Beans

[<<BACK] [CONTENTS] [NEXT>>]

Printable Page 

■ Requires login

Early Access ■
 Downloads

Bug Database ■
 Submit a Bug
 View Database

Newsletters
 Back Issues
 Subscribe

Learning Centers
 Articles
 Bookshelf
 Code Samples
 New to Java
 Question of the Week
 Quizzes
 Tech Tips
 Tutorials

Forums

The example uses two entity Beans and two session Beans. The entity Beans, `AuctionItemBean` and `RegistrationBean`, represent persistent items that could be stored in a database, and the session Beans, `SellerBean` and `BidderBean`, represent short-lived operations with the client and data.

The session Beans are the client interface to the entity beans. The `SellerBean` processes requests to add new auction items for sale. The `BidderBean` processes requests to retrieve auction items and place bids on those items. Changing and adding to the database data in a container-managed Bean is left to the entity Beans.

[Auction Servlet](#)

[Entity Beans](#)

[Session Beans](#)

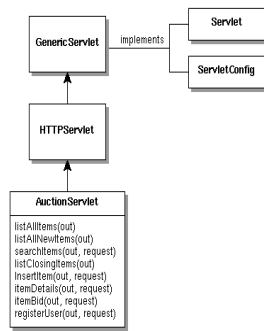
[Container Classes](#)

AuctionServlet

The [AuctionServlet](#) is essentially the second tier in the application and the focal point for auction activities. It accepts end user input from the browser by way of hypertext transfer protocol (HTTP), passes the input to the appropriate Enterprise Bean for processing, and displays the processed results to the end user in the browser.

Here is a [Unified Modeling Language \(UML\)](#) class diagram for the `AuctionServlet` class.

Technology Centers



The `AuctionServlet` methods shown above invoke business logic that executes on the server by looking up an Enterprise Bean and calling one or more of its methods. When the servlet adds HTML codes to a page for display to the user, that logic executes on the client.

For example, the `listAllItems(out)` method executes code on the client to dynamically generate an HTML page to be viewed by the client in a browser. The HTML page is populated with the results of a call to `BidderBean` that executes logic on the server to generate a list of all auction items.

```

private void listAllItems(ServletOutputStream out)
    throws IOException{

//Put text on HTML page
    setTitle(out, "Auction results");
    String text = "Click Item number for description
        and to place bid.";
    try{
        addLine("<BR>"+text, out);
//Look up Bidder bean home interface.
        BidderHome bhome=(BidderHome) ctx.lookup("bidder");
//Create Bidder bean remote interface.
        Bidder bid=bhome.create();
//Call Bidder bean method through remote interface.
        Enumeration enum=(Enumeration)bid.getItemList();
  
```

```

        if(enum != null) {
//Put retrieved items on servlet page.
            displayitems(enum, out);
            addLine("", out);
        }
    } catch (Exception e) {
//Print error on servlet page.
        addLine("AuctionServlet List All Items error",out);
        System.out.println("AuctionServlet <list>:"+e);
    }
    out.flush();
}

```

Entity Beans

AuctionItemBean and RegistrationBean are entity Beans.

AuctionItemBean adds new auction items to the database and updates the bid amount as users bid on the item. RegistrationBean adds information to the database on registered users. Both Beans consist of the classes described here.

AuctionItem Entity Bean

Here are the AuctionItemBean classes. Remember that these Enterprise Beans are distributed objects that use the Remote Method Invocation (RMI) API, so when an error occurs, an RMI remote exception is thrown.

[AuctionItem.java](#)

[AuctionItemHome.java](#)

[AuctionItemBean.java](#)

[AuctionItemPk.java](#)

AuctionItem is the remote interface. It describes what the Bean does by declaring the developer-defined methods that provide the business logic for this Bean. These methods are the ones used by the client to interact with the Bean over the remote connection. Its name maps to the AUCTIONITEMS table shown just below.

AuctionItemHome is the home interface. It describes how the Bean is created in, found in, and removed from its container. The Enterprise Bean server deployment tools will provide the implementation for this interface.

AuctionItemBean is the Enterprise Bean. It implements EntityBean, provides the business logic for the developer-defined methods, and implements EntityBean methods for creating the Bean and setting the session context. This is a class that the Bean developer needs to implement. Its field variables map to fields in the AUCTIONITEMS table shown just below.

`AuctionItemPK` is the primary key class. The Enterprise JavaBeans server requires a container-managed entity Bean to have a primary key class with a public primary key field (or fields, if using composite primary keys). The Bean developer implements this class. The `ID` field is the primary key in the `AUCTIONITEMS` table shown just below, so the `id` field is a public field in this class. The `id` field is assigned a value when the primary key class is constructed.

You can request the container manage database persistence for an Enterprise Bean or write the code to manage the persistence yourself. In this chapter, all beans (entity and session) are container-managed. With container-managed Beans, all you do is specify which fields are container managed and let the Enterprise JavaBeans server do the rest. This is great for simple applications, but if you are coding something that is fairly complex, you might need more control.

How to override the underlying Enterprise JavaBeans services to gain more control or provide similar services for non-Enterprise JavaBean applications is covered in Chapter 3.

Auction Items Table

Here is the `AUCTIONITEMS` table.

```
create table AUCTIONITEMS (SUMMARY VARCHAR(80) ,
ID INT ,
COUNTER INT ,
DESCRIPTION VARCHAR(1000) ,
STARTDATE DATE ,
ENDDATE DATE ,
STARTPRICE DOUBLE PRECISION ,
INCREMENT DOUBLE PRECISION ,
SELLER VARCHAR(30) ,
MAXBID DOUBLE PRECISION,
BIDCOUNT INT,
HIGHBIDDER VARCHAR(30) )
```

Registration Entity Bean

`RegistrationBean` consists of the same kinds of classes and database table as the `AuctionItem` Bean, except the actual business logic, database table fields, and primary key are somewhat different. Rather than describe the classes, you can browse them and refer back to the `AuctionItem` Bean discussion if you have questions.

[Registration.java](#)

[RegistrationHome.java](#)

[RegistrationBean.java](#)

[RegistrationPK.java](#)

Registration Table

Here is the REGISTRATION table.

```
create table REGISTRATION (THEUSER VARCHAR(40) ,  
PASSWORD VARCHAR(40) ,  
EMAILADDRESS VARCHAR(80) ,  
CREDITCARD VARCHAR(40) ,  
BALANCE DOUBLE PRECISION )
```

Session Beans

BidderBean and SellerBean are the session Beans. BidderBean retrieves lists of auction items, searches for an item, checks the user ID and password when someone places a bid, and stores new bids in the database. SellerBean checks the user ID and password when someone posts an auction item, and adds new auction items to the database.

Both session Beans are initially deployed as stateless Beans. A stateless Bean does not keep a record of what the client did in a previous call; whereas, a stateful Bean does. Stateful Beans are very useful if the operation is more than a simple lookup and the client operation depends on something that happened in a previous call.

Bidder Session Bean

Here are the BidderBean classes. Enterprise Beans use the Remote Method Invocation (RMI) API, so when an error occurs, an RMI remote exception is thrown.

There is no primary key class because these Beans are transient and no database access is involved. To retrieve auction items from the database, BidderBean creates an instance of AuctionItemBean, and to process bids, it creates an instance of RegistrationBean.

[Bidder.java](#)

[BidderHome.java](#)

[BidderBean.java](#)

[Auction.java](#)

Bidder is the remote interface. It describes what the Bean does by declaring the developer-defined methods that provide the

business logic for this Bean. These methods are the ones that the client calls remotely.

BidderHome is the home interface. It describes how the Bean is created in, found in, and removed from its container.

BidderBean is the Enterprise Bean. It implements `SessionBean`, provides the business logic for the developer-defined methods, and implements `SessionBean` methods for creating the Bean and setting the session context.

`Auction.java` contains a small class that declares variables used by `BidderBean`.

Seller Session Bean

`SellerBean` consists of the same kinds of classes as `BidderBean`, except the business logic is different. Rather than describe the classes, you can browse them and refer back to the `BidderBean` discussion if you have questions.

[Seller.java](#)

[SellerHome.java](#)

[SellerBean.java](#)

Container Classes

The classes needed by the container to deploy an Enterprise Bean onto a particular Enterprise JavaBeans server are generated with a deployment tool. The classes include `_Stub.class` and `_Skel.class` classes that provide the RMI hooks on the client and server respectively.


These classes are used for marshaling (moving) data between the client program and the Enterprise JavaBeans server. In addition, implementation classes are created for the interfaces and deployment rules defined for our Bean.

The `Stub` object is installed on or downloaded to the client system and provides a local proxy object for the client. It implements the remote interfaces and transparently delegates all method calls across the network to the remote object.

The `Skel` object is installed on or downloaded to the server system and provides a local proxy object for the server. It unwraps data received over the network from the `Stub` object for processing by

the server.

[\[TOP\]](#)

Printable Page 

[This page was updated: 4-Jun-2001]

[Products & APIs](#) | [Developer Connection](#) | [Docs & Training](#) | [Online Support](#)
[Community Discussion](#) | [Industry News](#) | [Solutions Marketplace](#) | [Case Studies](#)

[Glossary](#) | [Feedback](#) | [A-Z Index](#)

For more information on Java technology
and other software from Sun Microsystems, call:
(800) 786-7638
Outside the U.S. and Canada, dial your country's
[AT&T Direct Access Number](#) first.



Copyright © 1995-2001 [Sun Microsystems, Inc.](#)
All Rights Reserved. [Terms of Use](#). [Privacy Policy](#).



[Products & APIs](#)
[Developer Connection](#)
[Docs & Training](#)
[Online Support](#)
[Community Discussion](#)
[Industry News](#)
[Solutions Marketplace](#)
[Case Studies](#)

[Training Index](#)

Printable Page 

■ Requires login

Early Access ■
[Downloads](#)

Bug Database ■
[Submit a Bug](#)
[View Database](#)

Newsletters
[Back Issues](#)
[Subscribe](#)

Learning Centers
[Articles](#)
[Bookshelf](#)
[Code Samples](#)
[New to Java](#)
[Question of the Week](#)
[Quizzes](#)
[Tech Tips](#)
[Tutorials](#)

Forums

Writing Advanced Applications

Chapter 2 Continued: Examining a Container-Managed Bean

[<<BACK] [CONTENTS] [NEXT>>]

This section walks through the [RegistrationBean.java](#) code to show how easy it is to have the container manage persistent data storage to an underlying medium such as a database (the default). Chapter 3 modifies `RegistrationBean` to use Bean-managed persistence to handle database access and manage transactions.

[Member Variables](#)
[Create Method](#)
[Entity Context Methods](#)
[Load Method](#)
[Store Method](#)
[Connection Pooling](#)
[Deployment Descriptor](#)

Member Variables

A container-managed environment needs clues about which variables are for persistent storage and which are not. In the Java™ programming language, the `transient` keyword indicates variables to not include when data in an object is serialized and written to persistent storage. In the `RegistrationBean.java` class, the `EntityContext` variable is marked `transient` to indicate that its data not be written to the underlying storage medium.

`EntityContext` data is not written to persistent storage because its purpose is to provide information on the container's runtime context. It, therefore, does not contain data on the registered user and should not be saved to the underlying storage medium. The other variables are declared `public` so the container in this example can discover them using the Reflection API.

```
protected transient EntityContext ctx;
```

Technology Centers

```
public String theuser, password, creditcard,
           emailaddress;
public double balance;
```

Create Method

The Bean's `ejbCreate` method is called by the container after the client program calls the `create` method on the `remote` interface and passes in the registration data. This method assigns the incoming values to the member variables that represent user data. The container handles storing and loading the data, and creating new entries in the underlying storage medium.

```
public RegistrationPK ejbCreate(String theuser,
                               String password,
                               String emailaddress,
                               String creditcard)
    throws CreateException, RemoteException {

    this.theuser=theuser;
    this.password=password;
    this.emailaddress=emailaddress;
    this.creditcard=creditcard;
    this.balance=0;
```

Entity Context Methods

An entity Bean has an associated `EntityContext` instance that gives the Bean access to container-managed runtime information such as the transaction context.

```
public void setEntityContext(
    javax.ejb.EntityContext ctx)
    throws RemoteException {
    this.ctx = ctx;
}

public void unsetEntityContext()
    throws RemoteException{
    ctx = null;
}
```

Load Method

The Bean's `ejbLoad` method is called by the container to load data from the underlying storage medium. This would be necessary when `BidderBean` or `SellerBean` need to check a user's ID or password against the stored values.

Note: Not all Bean objects are live at any one time. The Enterprise JavaBeans™ server might have a configurable

number of Beans that it keeps in memory.

This method is not implemented because the Enterprise JavaBeans container seamlessly loads the data from the underlying storage medium for you.

```
public void ejbLoad() throws RemoteException {}
```

Store Method

The Bean's `ejbStore` method is called by the container to save user data. This method is not implemented because the Enterprise JavaBeans container seamlessly stores the data to the underlying storage medium for you.

```
public void ejbStore() throws RemoteException {}
```

Connection Pooling

Loading data from and storing data to a database can take a lot of time and reduce an application's overall performance. To reduce database connection time, the BEA Weblogic server uses a JDBC™ connection pool to cache database connections so connections are always available when the application needs them.

However, you are not limited to the default JDBC connection pool. You can override the Bean-managed connection pooling behaviour and substitute your own. [Chapter 8: Performance Techniques](#) explains how.

Deployment Descriptor

The remaining configuration for a container-managed persistent Beans occurs at deployment time. The following is the text-based Deployment Descriptor used in a BEA Weblogic Enterprise JavaBeans server.

Text Deployment Descriptor

```
(environmentProperties
  (persistentStoreProperties
    persistentStoreType      jdbc
  (jdbc
    tableName                registration
    dbIsShared               false
    poolName                 ejbPool
```

```

        (attributeMap
          creditcard          creditcard
          emailaddress        emailaddress
          balance              balance
          password             password
          theuser              theuser
        ); end attributeMap
      ); end jdbc
    ); end persistentStoreProperties
  ); end environmentProperties

```

The deployment descriptor indicates that storage is a database whose connection is held in a JDBC™ connection pool called `ejbPool`. The `attributeMap` contains the Enterprise Bean variable on the left and the associated database field on the right.

XML Deployment Descriptor

In Enterprise JavaBeans 1.1, the deployment descriptor uses XML. The equivalent configuration in XML is as follows:

```

<persistence-type>Container</persistence-type>
<cmp-field><field-name>creditcard
  </field-name></cmp-field>
<cmp-field><field-name>emailaddress
  </field-name></cmp-field>
<cmp-field><field-name>balance
  </field-name></cmp-field>
<cmp-field><field-name>password
  </field-name></cmp-field>
<cmp-field><field-name>theuser
  </field-name></cmp-field>
<resource-ref>
<res-ref-name>registration</res-ref-name>
<res-type>javax.sql.DataSource</res-type>
<res-auth>Container</res-auth>
</resource-ref>

```

The container managed-fields here map directly to their counterpart names in the database table. The container resource authorization (`res-auth`) means the container handles the database login for the `REGISTRATION` table.

[\[TOP\]](#)

Printable Page 

[This page was updated: 4-Jun-2001]

[Products & APIs](#) | [Developer Connection](#) | [Docs & Training](#) | [Online Support](#)
[Community Discussion](#) | [Industry News](#) | [Solutions Marketplace](#) | [Case Studies](#)

[Glossary](#) | [Feedback](#) | [A-Z Index](#)

For more information on Java technology
and other software from Sun Microsystems, call:
(800) 786-7638
Outside the U.S. and Canada, dial your country's
[AT&T Direct Access Number](#) first.



Copyright © 1995-2001 [Sun Microsystems, Inc.](#)
All Rights Reserved. [Terms of Use.](#) [Privacy Policy.](#)



[Products & APIs](#)
[Developer Connection](#)
[Docs & Training](#)
[Online Support](#)
[Community Discussion](#)
[Industry News](#)
[Solutions Marketplace](#)
[Case Studies](#)

[Training Index](#)

[Printable Page](#) 

■ Requires login

[Early Access](#) ■
[Downloads](#)

[Bug Database](#) ■
[Submit a Bug](#)
[View Database](#)

[Newsletters](#)
[Back Issues](#)
[Subscribe](#)

[Learning Centers](#)
[Articles](#)
[Bookshelf](#)
[Code Samples](#)
[New to Java](#)
[Question of the Week](#)
[Quizzes](#)
[Tech Tips](#)
[Tutorials](#)

[Forums](#)

Writing Advanced Applications

Chapter 2 Continued: Container-Managed finder Methods

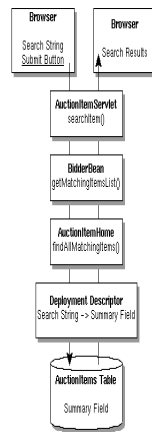
[<<BACK] [CONTENTS] [NEXT>>]

The auction house search facility is implemented as a container-managed `finder` method. It starts when the end user types in a search string and clicks the `Submit` button on the home page to locate an auction item. As shown in the diagram, the browser passes the search string to the `AuctionServlet.searchItem` method, which then passes it to the `BidderBean.getMatchingItemsList` method.

At this point, `BidderBean.getMatchingItemsList` passes the search string to the `findAllMatchingItems` method declared in the `AuctionItemHome` interface. This method is a `finder` method, and container implementations vary in how they handle calls to `finder` methods. [BEA Weblogic](#) containers look in the Bean's deployment descriptor for information on a Bean's `finder` methods.

In the case of the search, the deployment descriptor maps the search string passed to `AuctionItemHome.findAllMatchingItems` to the `summary` field in the underlying `AuctionItems` database table. This tells the Enterprise JavaBeans™ server to retrieve data for all auction items with a `summary` field that contains text that matches the search string.

Technology Centers



This section walks through the different parts of the `finder`-based search code. Chapter 3 describes how to create a Bean-managed search to handle complex queries and searches that span more than one Bean type (entity and session Beans) or database tables.

[AuctionServlet.searchItems](#)

[BidderBean.getMatchingItemsList](#)

[AuctionItemHome.findAllMatchingItems](#)

[AuctionItemBean Deployment Descriptor](#)

AuctionServlet.searchItems

The `searchItems` method retrieves the text string from the browser, creates an HTML page to display the search results, and passes the search string to the `BidderBean.getMatchingItemsList` method. `BidderBean` is a session Bean that retrieves lists of auction items and checks the user ID and password for end users seeking to bid on auction items.

The search results are returned to this method in an Enumeration variable.

```
private void searchItems(ServletOutputStream out,
    HttpServletRequest request)
    throws IOException {

    //Retrieve search string
    String searchString=request.getParameter(
        "searchString");

    //Create HTML page
    String text = "Click Item number for description
        and to place bid.";
    setTitle(out, "Search Results");
    try {
        addLine("<BR>"+text, out);

    //Look up home interface for BidderBean
        BidderHome bhome=(BidderHome) ctx.lookup(
            "bidder");

    //Create remote interface for BidderBean
        Bidder bid=bhome.create();

    //Pass search string to BidderBean method
        Enumeration enum=(Enumeration)
            bid.getMatchingItemsList(searchString);

        if(enum != null) {
            displayItems(enum, out);
            addLine("", out);
        }
    } catch (Exception e) {
        addLine("AuctionServlet Search Items error",
            out);
        System.out.println("AuctionServlet <newlist>:
            "+e);
    }
    out.flush();
}
```

BidderBean.getMatchingItemsList

The BidderBean.getMatchingItemsList method calls the AuctionItemHome.findAllMatchingItems method and passes it the search string. AuctionItemBean is an entity Bean that handles auction item updates and retrievals.

The search results are returned to this method in an Enumeration variable.

```
public Enumeration getMatchingItemsList(
    String searchString)
    throws RemoteException {

    Enumeration enum=null;
```

```

    try{
//Create Home interface for AuctionItemBean
        AuctionItemHome home = (AuctionItemHome)
            ctx.lookup("auctionitems");

//Pass search string to Home interface method
        enum=(Enumeration)home.findAllMatchingItems(
            searchString);
    }catch (Exception e) {
        System.out.println("getMatchingItemList: "+e);
        return null;
    }
    return enum;
}

```

AuctionItemHome.findAllMatchingItems

The `AuctionItemHome.findAllMatchingItems` method is not implemented in `AuctionItemBean`. The `AuctionItemBean` finder method implementations are defined in the `AuctionItemBean` deployment descriptor when [BEA Weblogic](#) containers are used.

When using these containers, even if the Bean has finder method implementations, they are ignored and the deployment descriptor settings are consulted instead.

```

//Declare method in Home interface
    public Enumeration findAllMatchingItems(
        String searchString)
        throws FinderException, RemoteException;

```

AuctionItemBean Deployment Descriptor

When a Bean's finder method is called, the container consults the deployment descriptor for that Bean to find out what data the finder method needs to retrieve from the underlying database table. The container passes this information to the Enterprise JavaBeans server, which does the actual retrieval.

The deployment descriptor for `AuctionItemBean` provides finderDescriptors for all finder methods declared in the `AuctionItemHome` interface. The finderDescriptor for the `findAllMatchingItems` method maps the search string to the summary field in the underlying `AuctionItems` database table. This tells the Enterprise JavaBeans server to retrieve the data for all table rows with a summary field that matches the text in the search string.


```

(finderDescriptors
    "findAllItems()"           "(= 1 1)"
    "findAllNewItems(java.sql.Date newtoday)"
        "(= startdate $newtoday)"
    "findAllClosedItems(java.sql.Date closedtoday)"

```

```
        "(= enddate $closedtoday)"  
    "findAllMatchingItems(String searchString)"  
        "(like summary $searchString)"  
); end finderDescriptors
```

[\[TOP\]](#)

Printable Page 

[This page was updated: 4-Jun-2001]

[Products & APIs](#) | [Developer Connection](#) | [Docs & Training](#) | [Online Support](#)
[Community Discussion](#) | [Industry News](#) | [Solutions Marketplace](#) | [Case Studies](#)

[Glossary](#) | [Feedback](#) | [A-Z Index](#)

For more information on Java technology
and other software from Sun Microsystems, call:

(800) 786-7638

Outside the U.S. and Canada, dial your country's
[AT&T Direct Access Number](#) first.



Copyright © 1995-2001 [Sun Microsystems, Inc.](#)
All Rights Reserved. [Terms of Use](#). [Privacy Policy](#).



[Products & APIs](#)
[Developer Connection](#)
[Docs & Training](#)
[Online Support](#)
[Community Discussion](#)
[Industry News](#)
[Solutions Marketplace](#)
[Case Studies](#)

[Training Index](#)

Printable Page 

■ Requires login

Early Access ■
[Downloads](#)

Bug Database ■
[Submit a Bug](#)
[View Database](#)

Newsletters
[Back Issues](#)
[Subscribe](#)

Learning Centers
[Articles](#)
[Bookshelf](#)
[Code Samples](#)
[New to Java](#)
[Question of the Week](#)
[Quizzes](#)
[Tech Tips](#)
[Tutorials](#)

Forums

Writing Advanced Applications

Chapter 3: Data and Transaction Management

[\[<<BACK\]](#) [\[CONTENTS\]](#) [\[NEXT>>\]](#)

When you use the Enterprise JavaBeans™ architecture, data is written to and read from the database without your writing any SQL code to do it. But what if you do not want to store the data in a database, or want to write your own SQL commands, or manage transactions? You can override the built-in container-managed persistence and implement Bean-managed persistence using your own data storage and transaction management code.

Bean-managed persistence comes in useful when you want more control than the container-managed persistence provides. For example you might want to override the default of most containers to map the data in one Bean to one row in a table, implement your own `finder` methods, or customize caching.

This chapter presents two versions of the `RegistrationBean` class from Chapter 2. One version reads user data from and writes it to a file using serialized input and output streams. The other version provides its own SQL commands for reading from and writing to the database. It also explains how you can write your own transaction management code.

[Bean-Managed Persistence and the JDBC™ Platform](#)
[Managing Transactions](#)
[Bean-Managed finder Methods](#)

In a Rush?


This table links you directly to specific topics.

Topic	Section

Technology Centers

Bean-Managed Persistence and the JDBC Platform	Connect to Database Create Method Load Method Refresh Method Store Method Find Method
Transaction Management	Why Manage Transactions? Session Synchronization Transaction Commit Mode
Bean-Managed finder Methods	AuctionServlet.searchItems SearchBean

[\[TOP\]](#)

Printable Page 

[This page was updated: 4-Jun-2001]

[Products & APIs](#) | [Developer Connection](#) | [Docs & Training](#) | [Online Support](#)
[Community Discussion](#) | [Industry News](#) | [Solutions Marketplace](#) | [Case Studies](#)

[Glossary](#) | [Feedback](#) | [A-Z Index](#)

For more information on Java technology and other software from Sun Microsystems, call: **(800) 786-7638**
 Outside the U.S. and Canada, dial your country's [AT&T Direct Access Number](#) first.



Copyright © 1995-2001 [Sun Microsystems, Inc.](#)
 All Rights Reserved. [Terms of Use](#). [Privacy Policy](#).



[Products & APIs](#)
[Developer Connection](#)
[Docs & Training](#)
[Online Support](#)
[Community Discussion](#)
[Industry News](#)
[Solutions Marketplace](#)
[Case Studies](#)

[Training Index](#)

[Printable Page](#) 

■ Requires login

[Early Access](#) ■

[Downloads](#)

[Bug Database](#) ■

[Submit a Bug](#)

[View Database](#)

[Newsletters](#)

[Back Issues](#)

[Subscribe](#)

[Learning Centers](#)

[Articles](#)

[Bookshelf](#)

[Code Samples](#)

[New to Java](#)

[Question of the Week](#)

[Quizzes](#)

[Tech Tips](#)

[Tutorials](#)

[Forums](#)

Writing Advanced Applications

Chapter 3 Continued: Bean-Managed Persistence and the JDBC™ Platform

[\[<<BACK\]](#) [\[CONTENTS\]](#) [\[NEXT>>\]](#)

There might be times when you want to override container-managed persistence and implement entity or session Bean methods to use the SQL commands you provide. This type of Bean-managed persistence can be useful if you need to improve performance or map data in multiple Beans to one row in a database table.

This section shows you how to convert the [RegistrationBean.java](#) class to access the database with the JDBC™ `PreparedStatement` class.

[Connect to Database](#)

[Create Method](#)

[Load Method](#)

[Refresh Method](#)

[Store Method](#)

[Find Method](#)

Connect to Database

This version of the [RegistrationBean.java](#) class establishes a connection to the database by instantiating a static `Driver` class and providing the `getConnection` method.

The `getConnection` method queries the static `DriverManager` class for a registered database driver that matches the Uniform Resource Locator (URL) . In this case, the URL is `weblogic.jdbc.jts.Driver`.

```
//Create static instance of database driver
static {
    new weblogic.jdbc.jts.Driver();
}
```

Technology Centers

```
//Get registered driver from static instance
public Connection getConnection() throws SQLException{
    return DriverManager.getConnection(
        "jdbc:weblogic:jts:ejbPool");
}
```

Create Method

The `ejbCreate` method assigns values to data member variables, gets a connection to the database, and creates an instance of the `java.sql.PreparedStatement` class to execute the SQL statement for writing the data to the `registration` table in the database.

A `PreparedStatement` object is created from a SQL statement which is sent to the database and precompiled before any data is sent. You call the appropriate `setXXX` statements on the `PreparedStatement` object to send the data. Keeping the `PreparedStatement` and `Connection` objects has private instance variables greatly reduces overhead because the SQL statement does not have to be compiled everytime data is sent.

The last thing the `ejbCreate` method does is create a primary key class with the user Id, and return it to the container.

```
public RegistrationPK ejbCreate(String theuser,
                                String password,
                                String emailaddress,
                                String creditcard)
    throws CreateException, RemoteException {

    this.theuser=theuser;
    this.password=password;
    this.emailaddress=emailaddress;
    this.creditcard=creditcard;
    this.balance=0;

    try {
        con=getConnection();
        ps=con.prepareStatement("insert into registration (
                                theuser, password,
                                emailaddress, creditcard,
                                balance) values (
                                ?, ?, ?, ?, ?)");
        ps.setString(1, theuser);
        ps.setString(2, password);
        ps.setString(3, emailaddress);
        ps.setString(4, creditcard);
        ps.setDouble(5, balance);
        if (ps.executeUpdate() != 1) {
            throw new CreateException (
                "JDBC did not create a row");
        }
        RegistrationPK primaryKey = new RegistrationPK();
        primaryKey.theuser = theuser;
    }
```

```

        return primaryKey;
    } catch (CreateException ce) {
        throw ce;
    } catch (SQLException sqe) {
        throw new CreateException (sqe.getMessage());
    } finally {
        try {
            ps.close();
        } catch (Exception ignore) {}
        try {
            con.close();
        } catch (Exception ignore) {}
    }
}

```

Load Method

This method gets the primary key from the entity context and passes it to the `refresh` method which loads the data.

```

public void.ejbLoad() throws RemoteException {
    try {
        refresh((RegistrationPK) ctx.getPrimaryKey());
    }
    catch (FinderException fe) {
        throw new RemoteException (fe.getMessage());
    }
}

```

Refresh Method

The `refresh` method is programmer-supplied code to load the data from the database. It checks the primary key value, gets a connection to the database, and creates a `PreparedStatement` object for querying the database for the user specified in the primary key.

Data is read from the database into a `ResultSet` and assigned to the global member variables so the `RegistrationBean` has the most up-to-date information for the user.

```

private void refresh(RegistrationPK pk)
    throws FinderException, RemoteException {

    if (pk == null) {
        throw new RemoteException ("primary key
            cannot be null");
    }
    Connection con = null;
    PreparedStatement ps = null;
    try {
        con=getConnection();
        ps=con.prepareStatement("select password,
            emailaddress, creditcard,
            balance from registration
            where theuser = ?");
    }
}

```

```

        ps.setString(1, pk.theuser);
        ps.executeQuery();
        ResultSet rs = ps.getResultSet();
        if (rs.next()) {
            theuser = pk.theuser;
            password = rs.getString(1);
            emailaddress = rs.getString(2);
            creditcard = rs.getString(3);
            balance = rs.getDouble(4);
        }
        else {
            throw new FinderException (
                "Refresh: Registration ("
                + pk.theuser + ") not found");
        }
    }
    catch (SQLException sqe) {
        throw new RemoteException (sqe.getMessage());
    }
    finally {
        try {
            ps.close();
        }
        catch (Exception ignore) {}
        try {
            con.close();
        }
        catch (Exception ignore) {}
    }
}
}

```

Store Method

This method gets a database connection and creates a PreparedStatement to update the database.

```

public void ejbStore() throws RemoteException {
    Connection con = null;
    PreparedStatement ps = null;
    try {
        con = getConnection();
        ps = con.prepareStatement("update registration
                                set password = ?,
                                emailaddress = ?,
                                creditcard = ?,
                                balance = ?
                                where theuser = ?");
        ps.setString(1, password);
        ps.setString(2, emailaddress);
        ps.setString(3, creditcard);
        ps.setDouble(4, balance);
        ps.setString(5, theuser);
        int i = ps.executeUpdate();
        if (i == 0) {
            throw new RemoteException (
                "ejbStore: Registration ("
                + theuser + ") not updated");
        }
    }
}

```

```

    } catch (RemoteException re) {
        throw re;
    } catch (SQLException sqe) {
        throw new RemoteException (sqe.getMessage());
    } finally {
        try {
            ps.close();
        } catch (Exception ignore) {}
        try {
            con.close();
        }
        catch (Exception ignore) {}
    }
}

```

Find Method

The `ejbFindByPrimaryKey` method matches the signature of the `FindByPrimaryKey` method in the [RegistrationHome](#) interface. It calls the `refresh` method to get or refresh the user data for the user specified by the primary key.

The container-managed persistence version of `RegistrationBean` does not implement this method because the container handles getting and refreshing the user data.


```

public RegistrationPK ejbFindByPrimaryKey(
    RegistrationPK pk)
    throws FinderException,
    RemoteException {

    if ((pk == null) || (pk.theuser == null)) {
        throw new FinderException ("primary key
            cannot be null");
    }
    refresh(pk);
    return pk;
}

```

[\[TOP\]](#)

Printable Page 

[This page was updated: 4-Jun-2001]

[Products & APIs](#) | [Developer Connection](#) | [Docs & Training](#) | [Online Support](#)
[Community Discussion](#) | [Industry News](#) | [Solutions Marketplace](#) | [Case Studies](#)

[Glossary](#) | [Feedback](#) | [A-Z Index](#)

For more information on Java technology
and other software from Sun Microsystems, call:
(800) 786-7638
Outside the U.S. and Canada, dial your country's
[AT&T Direct Access Number](#) first.



Copyright © 1995-2001 [Sun Microsystems, Inc.](#)
All Rights Reserved. [Terms of Use](#). [Privacy Policy](#).



[Products & APIs](#)
[Developer Connection](#)
[Docs & Training](#)
[Online Support](#)
[Community Discussion](#)
[Industry News](#)
[Solutions Marketplace](#)
[Case Studies](#)

[Training Index](#)

Writing Advanced Applications

Chapter 3 Continued: Managing Transactions

[<<BACK] [CONTENTS] [NEXT>>]

Printable Page 

■ Requires login

Early Access ■
[Downloads](#)

Bug Database ■
[Submit a Bug](#)
[View Database](#)

Newsletters
[Back Issues](#)
[Subscribe](#)

Learning Centers
[Articles](#)
[Bookshelf](#)
[Code Samples](#)
[New to Java](#)
[Question of the Week](#)
[Quizzes](#)
[Tech Tips](#)
[Tutorials](#)

Forums

Wouldn't it be great if every operation your application attempts succeeds? Unfortunately, in the multi-threaded world of distributed applications and shared resources, this is not always possible.

Why? First of all, shared resources must maintain a consistent view of the data to all users. This means reads and writes have to be managed so users do not overwrite each other's changes, or transaction errors do not corrupt data integrity. Also, if you factor in intermittent network delays or dropped connections, the potential for operations to fail in a web-based application increases as the number of users increases.

If operation failures are unavoidable, the next best thing is to recover safely, and that is where transaction management fits in. Modern databases and transaction managers let you undo and restore the state of a failed sequence of operations to ensure the data is consistent for access by multiple threads.

This section adds code to `SellerBean` from the auction house example so it can manage its auction item insertion transaction beyond the default transaction management provided by its container.

[Why Manage Transactions?](#)

[Session Synchronization](#)

- [Container-Managed Example](#)
- [Code](#)

[Transaction Commit Mode](#)

- [Server Configuration](#)
- [Transaction Attribute Descriptions](#)
- [Isolation Level Descriptions](#)
- [Bean-Managed Example](#)

Why Manage Transactions?

When you access a databases using the JDBC™ application programming interface (API), all operations are run with an explicit auto commit by default. This means any other application viewing this data will see the updated data after each JDBC call.

For simple applications this may be acceptable, but consider the auction application and the sequences that occur when `SellerBean` inserts an auction item. The user's account is first charged for listing the item, and the item is then added to the list of items up for auction. These operations involve `RegistrationBean` to debit the account and `AuctionItemBean` to add the item to the auction list.

In auto commit mode, if the auction item insertion fails, only the listing is backed out, and you have to manually adjust the user's account to refund the listing charge. In the meantime, another thread might try to deduct from the same user's account, find no credit left, and abort when perhaps a few milliseconds later it would have completed.

There are two ways to ensure the debit is backed out if the auction item insertion fails:

Add session synchronization code to a container-managed session Bean to gain control over transaction commits and roll backs.

Configure JDBC services to transaction commit mode and add code to start, stop, commit, and rollback the transaction. This is a Bean-managed transaction and can be used with an entity or session Bean.

Session Synchronization

A container-managed session Bean can optionally include session synchronization to manage the default auto commit provided by the container. Session synchronization code lets the container notify the Bean when important points in the transaction are reached. Upon receiving the notification, the Bean can take any needed actions before the transaction proceeds to the next point.

Note: A session Bean using Bean-managed transactions does not need session synchronization because it is in full control of the commit.

Container-Managed Example

`SellerBean` is a session Bean that uses `RegistrationBean` to check the user ID and password when someone posts an auction item and debit the seller's account for a listing, and `AuctionItemBean` to add new auction items to the database.

The transaction begins in the `insertItem` method with the account debit and ends when the entire transaction either commits or rolls back. The entire transaction including the 50 cents debit rolls back if the auction item is `null` (the insertion failed), or if an exception is caught. If the auction item is not `null` and the insertion succeeds, the entire transaction including the 50 cents debit commits.

Code

To use session synchronization, a session Bean implements the `SessionSynchronization` interface and its three methods, `afterBegin`, `beforeCompletion`, and `afterCompletion`. This example adapts the [SellerBean.java](#) code to use session synchronization.

```
public class SellerBean implements SessionBean,
    SessionSynchronization {

    private transient SessionContext ctx;
    private transient Properties p = new Properties();
    private transient boolean success = true;

    public void afterBegin() {}

    public void beforeCompletion() {
        if (!success) {
            ctx.setRollbackOnly();
        }
    }

    public void afterCompletion(boolean state) {}
}
```

afterBegin: The container calls this method before the debit to notify the session Bean a new transaction is about to begin. You can implement this method to do any preliminary database work that might be needed for the transaction. In this example, no preliminary database work is needed so this method has no implementation.

beforeCompletion: The container calls this method when it is ready to write the auction item and debit to the database, but before it actually does (commits). You can implement this method to write out any cached database updates or roll back the transaction. In

this example, the method calls the `setRollbackOnly` method on its session context in the event the `success` variable is set to `false` during the transaction.

afterCompletion: The container calls this method when the transaction commits. A `boolean` value of `true` means the data committed and `false` means the transaction rolled back. The method uses the `boolean` value to determine if it needs to reset the Bean's state in the case of a rollback. In this example, there is no need to reset the state in the event of a failure.

Here is the `insertItem` method with comments showing where the points where the `SessionSynchronization` methods are called.

```
public int insertItem(String seller,
                    String password,
                    String description,
                    int auctiondays,
                    double startprice,
                    String summary)
    throws RemoteException {

    try{
        Context jndiCtx = new InitialContext(p);

        RegistrationHome rhome =
            (RegistrationHome) sCtx.lookup("registration");
        RegistrationPK rpkm=new RegistrationPK();
        rpkm.theuser=seller;
        Registration newseller=rhome.findByPrimaryKey(rpkm);

        if((newseller == null) ||
            (!newseller.verifyPassword(password))) {
            return(Auction.INVALID_USER);
        }

        //Call to afterBegin
        newseller.adjustAccount(-0.50);

        AuctionItemHome home = (AuctionItemHome)
            jndiCtx.lookup("auctionitems");
        AuctionItem ai= home.create(seller,
                                   description,
                                   auctiondays,
                                   startprice,
                                   summary);

        if(ai == null) {
            success=false;
            return Auction.INVALID_ITEM;
        }
        else {
            return(ai.getId());
        }

    }catch(Exception e){
        System.out.println("insert problem="+e);
        success=false;
    }
}
```

```

        return Auction.INVALID_ITEM;
    }
    //Call to beforeCompletion
    //Call to afterCompletion
}

```

Transaction Commit Mode

If you configure the JDBC services to transaction commit mode, you can have the Bean manage the transaction. To set the JDBC services to commit, call `con.setAutoCommit(false)` on your JDBC connection. Not all JDBC drivers support commit mode, but to have the Bean control and manage transactions, you need a JDBC driver that does.

Transaction commit mode lets you add code that creates a safety net around a sequence of dependent operations. The Java™ Transaction API (JTA) provides the hooks you need to create that safety net. But, if you are using the Enterprise JavaBeans architecture, you can do it with a lot less code. You only have to configure the Enterprise JavaBeans server, and specify where the transaction starts, stops, rolls back, and commits in your code.

Server Configuration

Configuring the Enterprise JavaBeans server involves specifying the following settings in a configuration file for each Bean:

An isolation level to specify how exclusive a transaction's access to shared data is.

A transaction attribute to specify how to handle Bean-managed or container-managed transactions that continue in another Bean.

A transaction type to specify whether the transaction is managed by the container or the Bean.

For example, you would specify these settings for the [BEA Weblogic](#) server in a `DeploymentDescriptor.txt` file for each Bean.

Here is the part of the `DeploymentDescriptor.txt` for `SellerBean` that specifies the isolation level and transaction attribute. A description of the settings follows.

```

(controlDescriptors
  (DEFAULT

```

```

        isolationLevel          TRANSACTION_SERIALIZABLE
        transactionAttribute    REQUIRED
        runAsMode               CLIENT_IDENTITY
        runAsIdentity           guest
    ); end DEFAULT
); end controlDescriptors

```

Here is the equivalent Enterprise JavaBeans 1.1 extended markup language (XML) description that specifies the transaction type. In this example SellerBean is container managed.

```

<container-transaction>
  <method>
    <ejb-name>SellerBean</ejb-name>
    <method-name>*</method-name>
  </method>
  <transaction-type>Container</transaction-type>
  <trans-attribute>Required</trans-attribute>
</container-transaction>

```

In this example, SellerBean is Bean managed.

```

<container-transaction>
  <method>
    <ejb-name>SellerBean</ejb-name>
    <method-name>*</method-name>
  </method>
  <transaction-type>Bean</transaction-type>
  <trans-attribute>Required</trans-attribute>
</container-transaction>

```

Transaction Attribute Descriptions: An enterprise Bean uses a *transaction attribute* to specify whether a Bean's transactions are managed by the Bean itself or by the container, and how to handle transactions that started in another Bean.

The Enterprise JavaBeans server can control only one transaction at a time. This model follows the example set by the OMG Object Transaction Service (OTS), and means the current Enterprise JavaBeans specification does not provide a way to nest transactions. A nested transaction is a new transaction that starts from within an existing transaction. While transaction nesting is not allowed, continuing an existing transaction in another Bean is okay.

When a Bean is entered, the server creates a transaction context to manage the transaction. When the transaction is managed by the Bean, you access the context to begin, commit, and rollback the transaction as needed.

Here are the transaction attributes with a brief description for each one. The attribute names changed between the 1.0 and 1.1 versions of the Enterprise JavaBeans specification.

1.1 Specification

1.0 Specification

REQUIRED

TX_REQUIRED

Container-managed transaction. The server either starts and manages a new transaction on behalf of the user or continues using the transaction that was started by the code that called this Bean.

REQUIRESNEW

TX_REQUIRED_NEW

Container-managed transaction. The server starts and manages a new transaction. If an existing transaction starts this transaction, it suspends until this transaction completes.

Specified as Bean
transaction-type in
deployment descriptor

TX_BEAN_MANAGED

Bean-managed transaction. You access the transaction context to begin, commit, or rollback the transaction as needed.

SUPPORTS

TX_SUPPORTS

If the code calling this Bean has a transaction running, include this Bean in that transaction.

NEVER

TX_NOT_SUPPORTED

If the code calling a method in this Bean has a transaction running, suspend that transaction until the method called in this Bean completes. No transaction context is created for this Bean.

MANDATORY

TX_MANDATORY

The transaction attribute for this Bean is set when another Bean calls one of its methods. In this case, this Bean gets the transaction attribute of the calling Bean. If the calling Bean has no transaction attribute, the method called in this Bean throws a `TransactionRequired` exception.

Isolation Level Descriptions: An enterprise Bean uses an *isolation level* to negotiate its own interaction with shared data and the interaction of other threads with the same shared data. As the name implies, there are various levels of isolation with `TRANSACTION_SERIALIZABLE` providing the highest level of data integrity.

Note: Be sure to verify your database can handle the level you choose. In the Enterprise JavaBeans 1.1 specification, only Bean-managed persistence session Beans can set the isolation level.

If the database cannot handle the isolation level, the Enterprise JavaBeans server will get a failure when it tries to call the `setTransactionIsolation` JDBC method.

TRANSACTION_SERIALIZABLE: This level provides maximum data integrity. The Bean gets what amounts to exclusive access to the data. No other transaction can read or write this data until the serializable transaction completes.

Serializable in this context means *process as a serial operation*, and should not be confused with serializing objects to preserve and restore their states. Running transactions as a single serial operation is the slowest setting. If performance is an issue, use another isolation level that meets your application requirements, but provides better performance.

TRANSACTION_REPEATABLE_READ: At this level, data read by a transaction can be read, but not modified, by another transaction. The data is guaranteed to have the same value it had when first read, unless the first transaction changes it and writes the changed value back.

TRANSACTION_READ_COMMITTED: At this level, data read by a transaction cannot be read by other transactions until the first transaction either commits or rolls back.

TRANSACTION_READ_UNCOMMITTED: At this level, data involved in a transaction can be read by other threads before the first transaction either completes or rolls back. The other transactions cannot tell if the data was finally committed or rolled back.

Bean-Managed Example

`SellerBean` is a session Bean that uses `RegistrationBean` to check the user ID and password when someone posts an auction item and debit the seller's account for a listing, and `AuctionItemBean` to add new auction items to the database.

The transaction begins in the `insertItem` method with the account debit and ends when the entire transaction either commits or rolls back. The entire transaction including the 50 cents debit rolls back.

if the auction item is `null` (the insertion failed), or if an exception is caught. If the auction item is not `null` and the insertion succeeds, the entire transaction including the 50 cents debit commits.

For this example, the isolation level is `TRANSACTION_SERIALIZABLE`, and the transaction attribute is `TX_BEAN_MANAGED`. The other Beans in the transaction, `RegistrationBean` and `AuctionItemBean`, have an isolation level of `TRANSACTION_SERIALIZABLE` and a transaction attribute of `REQUIRED`.

Changes to this version of `SellerBean` over the container-managed version are flagged with comments.

```
public int insertItem(String seller,
                    String password,
                    String description,
                    int auctiondays,
                    double startprice,
                    String summary)
    throws RemoteException {

    //Declare transaction context variable using the
    //javax.transaction.UserTransaction class
    UserTransaction uts= null;

    try{
        Context ectx = new InitialContext(p);

    //Get the transaction context
        uts=(UserTransaction)ctx.getUserTransaction();

        RegistrationHome rhome = (
            RegistrationHome)ectx.lookup("registration");
        RegistrationPK rpkm=new RegistrationPK();
        rpkm.theuser=seller;
        Registration newseller=
            rhome.findByPrimaryKey(rpkm);

        if((newseller == null)||
            (!newseller.verifyPassword(password))) {
            return(Auction.INVALID_USER);
        }

    //Start the transaction
        uts.begin();


    //Deduct 50 cents from seller's account
        newseller.adjustAccount(-0.50);

        AuctionItemHome home = (
            AuctionItemHome) ectx.lookup("auctionitems");
        AuctionItem ai= home.create(seller,
            description,
            auctiondays,
            startprice,
            summary);
```

```
        if(ai == null) {
//Roll transaction back
            uts.rollback();
            return Auction.INVALID_ITEM;
        }
        else {
//Commit transaction
            uts.commit();
            return(ai.getId());
        }

    }catch(Exception e){
        System.out.println("insert problem="+e);
//Roll transaction back if insert fails
            uts.rollback();
            return Auction.INVALID_ITEM;
        }
    }
```

[\[TOP\]](#)

Printable Page 

[This page was updated: 4-Jun-2001]

[Products & APIs](#) | [Developer Connection](#) | [Docs & Training](#) | [Online Support](#)
[Community Discussion](#) | [Industry News](#) | [Solutions Marketplace](#) | [Case Studies](#)

[Glossary](#) | [Feedback](#) | [A-Z Index](#)

For more information on Java technology
and other software from Sun Microsystems, call:
(800) 786-7638
Outside the U.S. and Canada, dial your country's
[AT&T Direct Access Number](#) first.



Copyright © 1995-2001 [Sun Microsystems, Inc.](#)
All Rights Reserved. [Terms of Use](#). [Privacy Policy](#).



Products & APIs
 Developer Connection
 Docs & Training
 Online Support
 Community Discussion
 Industry News
 Solutions Marketplace
 Case Studies

[Training Index](#)

Printable Page

■ Requires login

Early Access ■
[Downloads](#)

Bug Database ■
[Submit a Bug](#)
[View Database](#)

Newsletters
[Back Issues](#)
[Subscribe](#)

Learning Centers
[Articles](#)
[Bookshelf](#)
[Code Samples](#)
[New to Java](#)
[Question of the Week](#)
[Quizzes](#)
[Tech Tips](#)
[Tutorials](#)

Forums

Writing Advanced Applications

Chapter 3 Continued: Bean-Managed finder Methods

[\[<<BACK\]](#) [\[CONTENTS\]](#) [\[NEXT>>\]](#)

The container-managed search described in Chapter 2 is based on a `finder` method mechanism where the deployment descriptor, rather than the Bean, specifies the `finder` method behavior. While the `finder` mechanism works well for simple queries and searches, it cannot handle complex operations that span more than one Bean type or database table. Also, the Enterprise JavaBeans™ 1.1 specification currently provides no specification for putting `finder` rules in the deployment descriptor.

So, for more complex queries and searches, you have to write Bean-managed queries and searches. This section explains how to write a Bean-managed version of the auction house search facility from Chapter 2. The Bean-managed search involves changes to the `AuctionServlet.searchItems` method and a new session Bean, `SearchBean`.

[AuctionServlet.searchItems](#)
[SearchBean](#)

AuctionServlet.searchItems

The search begins when the end user submits a search string to the search facility on the auction house home page, and clicks the `Submit` button. This invokes `AuctionServlet`, which retrieves the search string from the `HTTP` header and passes it to the `searchItem` method.

Note: The search logic for this example is fairly simple. The purpose is to show you how to move the search logic into a separate Enterprise Bean so you can create a more complex search on your own.

Technology Centers



The `searchItem` operation is in two parts: 1) Using the search string to retrieve primary keys, and 2) Using primary keys to retrieve auction items.

Part 1: The first thing the [searchItems](#) method does is pass the search string submitted by the end user to the `SearchBean` session Bean.

`SearchBean` (described in the next heading) implements a Bean-managed search that retrieves a list of primary keys for all auction items whose `Summary` fields contain characters matching the search string. This list is returned to the `searchItems` method in an `Enumeration` variable.

```
Enumeration enum=(Enumeration)
    search.getMatchingItemsList(searchString);
```

Part 2: The `searchItems` method then uses the returned `Enumeration` list from Part 1 and [AuctionItemBean](#) to retrieve each Bean in turn by calling `findByPrimaryKey` on each primary key in the list. This is a container-managed search based on the `finder` mechanism described in Chapter 2.



```
//Iterate through search results
while ((enum != null) &&
    enum.hasMoreElements())) {
    while(enum.hasMoreElements(in)) {

//Locate auction items
    AuctionItem ai=ahome.findByPrimaryKey((
        AuctionItemPK)enum.nextElement());
    displayLineItem(ai, out);
    }
}
```

SearchBean

The [SearchBean.java](#) class defines a Bean-managed search for the primary keys of auction items with `summary` fields that contain characters matching the search string. This Bean establishes a database connection, and provides the `getMatchingItemsList` and `EJBCreate` methods.

Database Connection

Because this Bean manages its own database access and search, it has to establish its own database connection. It cannot rely on the container to do this.

The database connection is established by instantiating a static Driver class and providing the getConnection method. The getConnection method queries the static DriverManager class for a registered database driver that matches the Uniform Resource Locator (URL) . In this case, the URL is weblogic.jdbc.jts.Driver.

```
//Establish database connection
static {
    new weblogic.jdbc.jts.Driver();
}

public Connection getConnection()
    throws SQLException {
    return DriverManager.getConnection(
        "jdbc:weblogic:jts:ejbPool");
}
```

Get Matching Items List

The getMatchingItemsList method looks up AuctionItemsBean and creates a PreparedStatement object for querying the database for summary fields that contain the search string. Data is read from the database into a ResultSet, stored in a Vector, and returned to AuctionServlet.

```
public Enumeration getMatchingItemsList(
    String searchString)
    throws RemoteException {

    ResultSet rs = null;
    PreparedStatement ps = null;
    Vector v = new Vector();
    Connection con = null;

    try{
//Get database connection
        con=getConnection();
//Create a prepared statement for database query
        ps=con.prepareStatement("select id from
            auctionitems where summary like ?");
        ps.setString(1, "%"+searchString+"%");
//Execute database query
        ps.executeQuery();
//Get results set
        rs = ps.getResultSet();
//Get information from results set
        AuctionItemPK pk;
        while (rs.next()) {
            pk = new AuctionItemPK();
            pk.id = (int)rs.getInt(1);
//Store retrieved data in vector
            v.addElement(pk);
        }
    }
}
```

```

    }
    rs.close();
    return v.elements();

} catch (Exception e) {
    System.out.println("getMatchingItemsList:
        "+e);
    return null;
}finally {
    try {
        if(rs != null) {
            rs.close();
        }
        if(ps != null) {
            ps.close();
        }
        if(con != null) {
            con.close();
        }
    } catch (Exception ignore) {}
}
}

```

Create Method


The `ejbCreate` method creates an `javax.naming.InitialContext` object. This is a Java™ Naming and Directory (JNDI) class that lets `SearchBean` access the database without relying on the container.

```

public void ejbCreate() throws CreateException,
    RemoteException {
    Properties p = new Properties();
    p.put(Context.INITIAL_CONTEXT_FACTORY,
        "weblogic.jndi.TengahInitialContextFactory");
    try{
        ctx = new InitialContext(p);
    }catch(Exception e) {
        System.out.println("create exception: "+e);
    }
}

```

[\[TOP\]](#)

Printable Page 

[This page was updated: 4-Jun-2001]

[Products & APIs](#) | [Developer Connection](#) | [Docs & Training](#) | [Online Support](#)
[Community Discussion](#) | [Industry News](#) | [Solutions Marketplace](#) | [Case Studies](#)

[Glossary](#) | [Feedback](#) | [A-Z Index](#)

For more information on Java technology
and other software from Sun Microsystems, call:
(800) 786-7638
Outside the U.S. and Canada, dial your country's
[AT&T Direct Access Number](#) first.



Copyright © 1995-2001 [Sun Microsystems, Inc.](#)
All Rights Reserved. [Terms of Use](#). [Privacy Policy](#).



[Products & APIs](#)
[Developer Connection](#)
[Docs & Training](#)
[Online Support](#)
[Community Discussion](#)
[Industry News](#)
[Solutions Marketplace](#)
[Case Studies](#)

[Training Index](#)

Writing Advanced Applications

Chapter 4: Distributed Computing

[\[<<BACK\]](#)
[\[CONTENTS\]](#)
[\[NEXT>>\]](#)

Printable Page 

■ Requires login

Early Access ■
[Downloads](#)

Bug Database ■
[Submit a Bug](#)
[View Database](#)

Newsletters
[Back Issues](#)
[Subscribe](#)

Learning Centers
[Articles](#)
[Bookshelf](#)
[Code Samples](#)
[New to Java](#)
[Question of the Week](#)
[Quizzes](#)
[Tech Tips](#)
[Tutorials](#)

Forums

As recently as ten years ago, distributed computing generally meant you had client PCs in one room with a server in another room. The problem with this architecture is if the connection to the server is lost, clients cannot update the payroll, sales, or other distributed company databases.

To prevent this sort of down time, different networking models were created. One example is the master and slave server model where if the master fails, the slaves take over. The problem with the different networking models is they all required some form of manual intervention and were tied to one operating system or language. And while these approaches met some of the short-term requirements for decreasing down time, they did not apply to heterogeneous distributed systems consisting of mixed network protocols and machines.

The Java™ platform combined with other advances such as Common Object Request Broker Architecture (CORBA), multi-tiered servers, and wireless networks has brought the realization of fully distributed computing a step further from the traditional client and server approach.

Now you can build applications that include service redundancy by default. If one server connection fails, you can seamlessly use a service on another server. CORBA and Distributed Component Object Model (DCOM) bridges mean that objects can be transferred between virtually all machines and languages. And with the new Jini™ System software, the distributed computing environment can soon be part of everything in your home, office or school. In short, distributed computing has never before been as important as it is today.

[Lookup Services](#)

Technology Centers

[Remote Method Invocation \(RMI\)](#)[Common Object Request Broker Architecture \(CORBA\)](#)[JDBC™ Technology](#)[Servlets](#)

In a Rush?

This table links you directly to specific topics.

Topic	Section
Lookup Services	Java Naming and Directory Interface (JNDI) Common Object Request Broker Architecture (CORBA) Naming Service Interoperable Object References (IOR) Remote Method Invocation (RMI) RMI Over Internet Inter-ORB Protocol (IIOP) JINI Lookup Services Improving Lookup Performance
Remote Method Invocation (RMI)	About RMI RMI in the Auction Application <ul style="list-style-type: none"> • Class Overview • File Summary • Compile the Example • Start the RMI Registry • Start the Remote Server Establishing Remote Communications RegistrationServer Class <ul style="list-style-type: none"> • Exporting a Remote Object • Passing by Value and by Reference • Distributed Garbage Collection Registration Interface ReturnResults Interface SellerBean Class

<p>Common Object Request Broker Architecture (CORBA)</p>	<p>IDL Mapping Scheme</p> <ul style="list-style-type: none"> • Quick Reference • Setting up IDL Mappings • Other IDL Types <p>CORBA in the Auction Application</p> <ul style="list-style-type: none"> • CORBA RegistrationServer • IDL Mappings File • Compiling the IDL Mappings File • Stub and Skeleton Files <p>Object Request Broker (ORB)</p> <ul style="list-style-type: none"> • Making the CORBA Server Accessible • Plugging in a New ORB • Naming Service Access by CORBA Clients <p>Helper and Holder Classes</p> <p>Garbage Collection</p> <p>CORBA Callbacks</p> <p>Using the Any Type</p> <p>Conclusion</p>
<p>JDBC Technology</p>	<p>JDBC Drivers</p> <p>Database Connections</p> <p>Statements</p> <ul style="list-style-type: none"> • Callable Statements • Statements • Prepared Statements <p>Caching Database Results</p> <p>Result Sets</p> <p>Scrolling Result Sets</p> <p>Controlling Transactions</p> <p>Escaping Characters</p> <p>Mapping Database Types</p> <p>Mapping Date types</p>

[\[TOP\]](#)

Printable Page 

[This page was updated: 4-Jun-2001]

[Products & APIs](#) | [Developer Connection](#) | [Docs & Training](#) | [Online Support](#)
[Community Discussion](#) | [Industry News](#) | [Solutions Marketplace](#) | [Case Studies](#)

[Glossary](#) | [Feedback](#) | [A-Z Index](#)

For more information on Java technology
and other software from Sun Microsystems, call:
(800) 786-7638
Outside the U.S. and Canada, dial your country's
[AT&T Direct Access Number](#) first.



Copyright © 1995-2001 [Sun Microsystems, Inc.](#)
All Rights Reserved. [Terms of Use](#). [Privacy Policy](#).



[Products & APIs](#)
[Developer Connection](#)
[Docs & Training](#)
[Online Support](#)
[Community Discussion](#)
[Industry News](#)
[Solutions Marketplace](#)
[Case Studies](#)

[Training Index](#)

Writing Advanced Applications

Chapter 4: Lookup Services

[\[<<BACK\]](#)
[\[CONTENTS\]](#)
[\[NEXT>>\]](#)

[Printable Page](#) 

Lookup services enable communications over a network. A client program can use a lookup protocol to get information on remote programs or machines and use that information to establish a communication.

■ Requires login

One common lookup service you might already be familiar with is Directory Name Service (DNS). It maps Internet Protocol (IP) addresses to machine names. Programs use the DNS mapping to look up the IP address associated with a machine name and use the IP address to establish a communication.

[Early Access](#) ■
[Downloads](#)

[Bug Database](#) ■
[Submit a Bug](#)
[View Database](#)

In the same way, the [AuctionServlet](#) presented in [Chapter 2](#) uses the naming service built into the Enterprise JavaBeans™ architecture to look up and reference Enterprise Beans registered with the Enterprise JavaBeans™ server.

[Newsletters](#)
[Back Issues](#)
[Subscribe](#)

[Learning Centers](#)
[Articles](#)
[Bookshelf](#)
[Code Samples](#)
[New to Java](#)
[Question of the Week](#)
[Quizzes](#)
[Tech Tips](#)
[Tutorials](#)

In addition to naming services, some lookup protocols provide directory services. Directory services such as Lightweight Directory Access Protocol (LDAP) and Sun's NIS+ provide other information and services beyond what is available with simple naming services. For example, NIS+ associates a `workgroup` attribute with a user account. This attribute can be used to restrict access to a machine so only the users in the specified `workgroup` have access.

[Forums](#)

This chapter describes how the Java™ Naming and Directory Interface (JNDI) is used in the auction application to look up Enterprise Beans. It also explains how to use some of the many other lookup services that have become available over time. The code to use these other services is not as simple as the lookup code in the auction application in Chapter 2, but the advantages to these other services can outweigh the need for more complex code in some situations.

Technology Centers

[Java Naming and Directory Interface \(JNDI\)](#)
[Common Object Request Broker Architecture \(CORBA\)](#)
[Naming Service](#)
[Interoperable Object References \(IOR\)](#)
[Remote Method Invocation \(RMI\)](#)
[RMI Over Internet Inter-ORB Protocol \(IIOP\)](#)
[JNI Lookup Services](#)
[Improving Lookup Performance](#)

Java Naming and Directory Interface (JNDI)

The JNDI application programming interface (API) makes it easy to plug lookup services from various providers into a program written in the Java language. As long as the client and server both use the same lookup service, the client can easily look up information registered with the server and establish communication.

The auction application session Beans use JNDI and a special JNDI naming factory from BEA Weblogic to look up entity Beans. JNDI services normally initialize the naming factory as a property on the command line or as an initialization value.

First, the naming factory `weblogic.jndi.TengahInitialContextFactory` is put into a `java.util.Property` object, then the `Property` object is passed as a parameter to the `InitialContext` constructor. Here is an example `ejbCreate` method.

```

Context ctx; //JNDI context

public void ejbCreate()
    throws CreateException, RemoteException {
    Hashtable env = new Hashtable();
    env.put(Context.INITIAL_CONTEXT_FACTORY,
        "weblogic.jndi.TengahInitialContextFactory");
    try{
        ctx = new InitialContext(env);
    }catch(Exception e) {
        System.out.println("create exception: "+e);
    }
}

```

Once created, the JNDI context is used to look up Enterprise Bean home interfaces. In this example, a reference to the Enterprise Bean bound to the name `registration` is retrieved and used for further operations.

```

RegistrationHome rhome =
    (RegistrationHome) ctx.lookup("registration");
RegistrationPK rpck=new RegistrationPK();
rpck.theuser=buyer;

```

```
Registration newbidder =
    rhome.findByPrimaryKey(rp);
```

On the server side, the deployment descriptor for the `RegistrationBean` has its `beanhomeName` value set to `registration`. Enterprise JavaBeans tools generate the rest of the naming code for the server.

The server calls `ctx.bind` to bind the name `registration` to the JNDI context. The `this` parameter references the `_stub` class that represents the `RegistrationBean`.

```
ctx.bind("registration", this);
```

JNDI is not the only way to look up remote objects. Lookup services are also available in the RMI, JINI, and CORBA platforms. You can use these platform-specific lookup services directly or from the JNDI API. JNDI allows the application to change the name service with little effort. For example, here are the code changes to have the `BidderBean.ejbCreate` method use the `org.omb.CORBA` lookup services instead of the default BEA Weblogic lookup services.

```
Hashtable env = new Hashtable();
env.put("java.naming.factory.initial",
    "com.sun.jndi.cosnaming.CNCtxFactory");
Context ic = new InitialContext(env);
```

CORBA Naming Service

The Common Object Request Broker Architecture (CORBA) defines a specification for objects in a distributed system to communicate with each other. Objects that use the CORBA specification to communicate are called CORBA objects, and consist of client and server objects.

CORBA objects can be written in any language with Interface Definition Language (IDL) mapping. These languages include the Java programming language, C++, and many traditional non-object-orientated languages.

The naming lookup service, like all other CORBA specifications, is defined in terms of IDL. The IDL module for the CORBA lookup service is called `CosNaming`. Any platform with an IDL mapping, such as the `idltojava` tool, can use this service to look up and discover CORBA objects. The IDL module for the CORBA lookup service is available in the Java 2 platform in the `org.omg.CosNaming` package.

The key interface in the `CosNaming` module is `NamingContext`. The `NamingContext` interface defines methods to bind objects to a name, list those bidding, and retrieve bound object references.



In addition to these public interfaces are helper classes. The `NameComponent` helper class is used in CORBA client and server programs to build the full name for the object reference name. The full name is an array of one or more `NameComponents` that indicates where to find the objects. The

naming scheme can be application specific.

For example in the auction application, the full name can be defined to use `auction` as the root naming context, and `RegistrationBean` and `AuctionItemBean` as children of the root context. This in effect employs a similar naming scheme as that used for the application class packaging.

In this example, the auction application has adapted `SellerBean` to a CORBA naming service to look up the CORBA `RegistrationBean`. The following code is extracted from the `SellerBean`, which acts as the CORBA client, the and `RegistrationServer` CORBA server.

CORBA RegistrationServer

This code in the [RegistrationServer](#) program creates a `NameComponent` object that indicates where to locate the `RegistrationBean` using `auction` and `RegistrationBean` as the full name.

```

NameComponent[] fullname = new NameComponent[2];
fullname[0] = new NameComponent("auction", "");
fullname[1] = new NameComponent(
    "RegistrationBean", "");
  
```

This next code binds the `fullname` as a new context. The first elements in the full name (`auction` in this example) are placeholders for building the context naming tree. The last element of the full name (`RegistrationBean` in this example) is the name submitted as the binding to the object.

```

String[] orbargs = { "-ORBInitialPort 1050"};
ORB orb = ORB.init(orbargs, null) ;

RegistrationServer rs= new RegistrationServer();
orb.connect(rs);

try{
    org.omg.CORBA.Object nameServiceObj =
        orb.resolve_initial_references("NameService");
  
```

```

NamingContext nctx =
    NamingContextHelper.narrow(nameServiceObj);
NameComponent[] fullname = new NameComponent[2];
fullname[0] = new NameComponent("auction", "");
fullname[1] = new NameComponent(
    "RegistrationBean", "");

NameComponent[] tempComponent =
    new NameComponent[1];
for(int i=0; i < fullname.length-1; i++ ) {
    tempComponent[0]= fullname[i];
    try{
        nctx=nctx.bind_new_context(tempComponent);
    }catch (Exception e){}
}
tempComponent[0]=fullname[fullname.length-1];

// finally bind the object to the full context path
nctx.bind(tempComponent, rs);

```

Once the `RegistrationServer` object is bound, it can be looked up with a JNDI lookup using a `CosNaming` service provider as described at the end of the section on JNDI, or using the CORBA name lookup service. Either way, the CORBA name server must be started before any look ups can happen. In the Java 2 platform, the CORBA nameserver is started as follows:

```
tnameserv
```

This starts the CORBA `RegistrationServer` on the default TCP port 900. If you need to use a different port, you can start the server like this. On Unix systems only root can access port numbers lower than 1025,

```
tnameserv -ORBInitialPort 1091
```

CORBA SellerBean

On the client side, the CORBA lookup uses the `NameComponent` object to construct the name. Start the object server as follows:

```
java registration.RegistrationServer
```

The difference in the client is that this name is passed to the `resolve` method which returns the CORBA object. The following code from the [SellerBean](#) object illustrates this point.

```

String[] args = { "-ORBInitialPort 1050"};
orb = ORB.init(args, null) ;
org.omg.CORBA.Object nameServiceObj =
    orb.resolve_initial_references("NameService") ;
nctx= NamingContextHelper.narrow(nameServiceObj);

NameComponent[] fullname = new NameComponent[2];
fullname[0] = new NameComponent("auction", "");
fullname[1] = new NameComponent(
    "RegistrationBean", "");

org.omg.CORBA.Object cobject= nctx.resolve(fullname);

```

The `narrow` method, from the `ObjectHelper` method, is generated by the IDL compiler, which provides a detailed mapping to translate each CORBA field into its respective Java language field. For example, the `SellerBean.insertItem` method looks up a registration CORBA object using the name `RegistrationBean`, and returns a `RegistrationHome` object. With the `RegistrationHome` object, you can return a `Registration` record by calling its `findByPrimaryKey` method.

```
org.omg.CORBA.Object cobject= nctx.resolve(fullname);
RegistrationHome regHome=
RegistrationHomeHelper.narrow(cobject);
RegistrationHome regRef =
    RegistrationHomeHelper.narrow(
        nctx.resolve(fullname));
RegistrationPKImpl rpk= new RegistrationPKImpl();
rpk.theuser(seller);
Registration newseller =
    RegistrationHelper.narrow(
        regRef.findByPrimaryKey(rpk));
if((newseller == null)||
    (!newseller.verifyPassword(password))) {
    return(Auction.INVALID_USER);
}
```

Interoperable Object References (IOR)

Using a CORBA name service works for most of CORBA applications especially when the object request brokers (ORBs) are supplied by one vendor. However, you might find the name service is not completely compatible among all ORBs, and you could get a frustrating `COMM_FAILURE` message when the CORBA client tries to connect to the CORBA server.

The solution is to use an Interoperable Object Reference (IOR) instead. An IOR is available in ORBs that support the Internet Inter-ORB protocol (IIOP). It contains the information that a naming service would keep for each object such as the host and port where the object resides, a unique lookup key for the object on that host, and what version of IIOP is supported.

IOR Server

To create an IOR all you do is call the `object_to_string` method from the `ORB` class and pass it an instance of the object. For example, to convert the `RegistrationServer` object to an IOR, you need to add the line `String ref = orb.object_to_string(rs);` to the following code in the `main` program:

```
String[] orbargs= {"-ORBInitialPort 1050"};
ORB orb = ORB.init(orbargs, null);
```

```

    RegistrationServer rs = new RegistrationServer();
//Add this line
    String ref = orb.object_to_string(rs);

```

So, instead of retrieving this object information from a naming service, there is another way for the server to send information to the client. You can register the returned `String` with a substitute name server, which can be a simple HTTP web server because the object is already in a transmittable format.

IOR Client

This example uses an HTTP connection to convert the IOR string back into an object. You call the `string_to_object` method from the ORB class. This method requests the IOR from the `RegistrationServer` and returns the IOR string. The `String` is passed to the ORB using the `ORB.string_to_object` method, and the ORB returns the remote object reference:

```

URL iorserver = new URL(
    "http://server.com/servlet?object=registration");
URLConnection con = iorserver.openConnection();
BufferedReader br = new BufferedReader(
    new InputStreamReader(con.getInputStream()));
String ref = br.readLine();
org.omg.CORBA.Object cobj = orb.string_to_object(ref);
RegistrationHome regHome =
    RegistrationHomeHelper.narrow(cobj);

```

The substitute name server can keep persistent IOR records that can survive a restart if needed.

Remote Method Invocation (RMI)

The Remote Method Invocation (RMI) API originally used its own communication protocol called Java Remote Method Protocol (JRMP), which resulted in having its own lookup service. Newer releases of RMI can now use the more ubiquitous IIOP protocol, in addition to JRMP. RMI-IIOP is covered in the next section.

The JRMP RMI naming service is similar to other lookup and naming services. The actual lookup is achieved by calling `Naming.lookup` and passing a URL parameter to that method. The URL specifies the machine name, an optional port where the RMI naming server, `rmiregistry`, that knows about that object is running, and the remote object you want to reference and call methods on.

For example:

```

SellerHome shome =
    (SellerHome)Naming.lookup(

```

```
"rmi://appserver:1090/seller");
```

This code returns the remote `SellerHome` reference `_stub` from the object bound to the name `seller` on the machine called `appserver`. The `rmi` part of the URL is optional and you may have seen RMI URLs without it, but if you are using JNDI or RMI-IIOP, including `rmi` in the URL will save confusion later on. Once you have a reference to `SellerHome`, you can call its methods.

In contrast to the JNDI lookup performed by `AuctionServlet.java`, which requires a two-stage lookup to create a context and then the actual lookup, RMI initializes the connection to the RMI name server, `rmiregistry`, and also gets the remote reference with one call.

This remote reference is leased to the client from the `rmiregistry`. The lease means that unless the client informs the server it still needs a reference to the object, the lease expires and the memory is reclaimed. This leasing operation is transparent to the user, but can be tuned by setting the server property `java.rmi.dgc.leaseValue` value in milliseconds when starting the server as follows:

```
java -Djava.rmi.dgc.leaseValue=120000 myAppServer
```

RMI Over Internet Inter-ORB Protocol (IIOP)

The advent of RMI over Internet Inter-ORB Protocol (IIOP), means existing RMI code can reference and look up an object with the CORBA `CosNaming` service. This gives you greater interoperability between architectures with little change to your existing RMI code.

Note: The `rmic` compiler provides the `-iiop` option to generate the stub and tie classes necessary for RMI-IIOP.

IIOP Server

The RMI-IIOP protocol is implemented as a JNDI plug-in, so as before, you need to create an `InitialContext`:

```
Hashtable env = new Hashtable();
env.put("java.naming.factory.initial",
        "com.sun.jndi.cosnaming.CNCTXFactory");
env.put("java.naming.provider.url",
        "iiop://localhost:1091");
Context ic = new InitialContext(env);
```

The naming factory should look familiar as it is the same CORBA

naming service used in the CORBA section. The main difference is the addition of a URL value specifying the naming service to which to connect. The naming service used here is the `tnameserv` program started on port 1091.

```
tnameserv -ORBInitialPort 1091
```

The other main change to the server side is to replace calls to `Naming.rebind` to use the JNDI `rebind` method in the `InitialContext` instance. For example:

Old RMI lookup code:

```
SellerHome shome= new SellerHome("seller");
Naming.rebind("seller", shome);
```

New RMI code:

```
Hashtable env = new Hashtable();
env.put("java.naming.factory.initial",
        "com.sun.jndi.cosnaming.CNCtxFactory");
env.put("java.naming.provider.url",
        "iiop://localhost:1091");
Context ic = new InitialContext(env);

SellerHome shome= new SellerHome("seller");
ic.rebind("seller", shome);
```

IIOP Client

On the client side, the RMI lookup is changed to use an instance of the `InitialContext` in the place of RMI `Naming.lookup`. The return object is mapped to the requested object by using the `narrow` method of the `javax.rmi.PortableRemoteObject` class.

`PortableRemoteObject` replaces `UnicastRemoteObject` that was previously available in the RMI server code.

Old RMI code:

```
SellerHome shome=(SellerHome)Naming.lookup(
    "rmi://appserver:1090/seller");
```

New RMI code:

```
Hashtable env = new Hashtable();
env.put("java.naming.factory.initial",
        "com.sun.jndi.cosnaming.CNCtxFactory");
env.put("java.naming.provider.url",
        "iiop://localhost:1091");
Context ic = new InitialContext(env);

SellerHome shome=
    (SellerHome)PortableRemoteObject.narrow(
        ic.lookup("seller"), SellerHome)
```

The `PortableRemoteObject` replaces `UnicastRemoteObject` previously

available in the RMI server code. The RMI code would either **extend** `UnicastRemoteObject` or call the `exportObject` method from the `UnicastRemoteObject` class. The `PortableRemoteObject` also contains an equivalent `exportObject` method. In the current implementation, it is best to explicitly remove unused objects by calling `PortableRemoteObject.unexportObject()`.

JINI lookup services

(To be done later)

Improving Lookup Performance

When you run your application, if you find it would be faster to walk the object to the other computer on a floppy, you have a network configuration problem. The source of the problem is how host names and IP addresses are resolved, and there is a workaround.

RMI and other naming services use the `InetAddress` class to obtain resolved host name and IP addresses. `InetAddress` caches lookup results to improve subsequent calls, but when it is passed a new IP address or host name, it performs a cross-reference between the IP address and the host name to prevent address spoofing. If you supply the host name as an IP address, `InetAddress` still tries to verify the name of the host.

To workaround this problem, include the host name and IP address in a hosts file on the client.

Unix Systems: On Unix, the hosts file is usually `/etc/hosts`.

Windows: On Windows 95 or 98, the hosts file is `c:\windows\hosts`, (the `hosts.sam` file is a sample file). On Windows NT, the hosts file is `c:\winnt\system32\drivers\etc\hosts`

All you need to do is put these lines in your hosts file. The `myserver1` and `myserver2` entries are the hosts running the remote server and `rmiregistry`

```
127.0.0.1 localhost
129.1.1.1 myserver1
129.1.1.2 myserver2
```

[\[TOP\]](#)

Printable Page 

[This page was updated: 4-Jun-2001]

[Products & APIs](#) | [Developer Connection](#) | [Docs & Training](#) | [Online Support](#)
[Community Discussion](#) | [Industry News](#) | [Solutions Marketplace](#) | [Case Studies](#)

[Glossary](#) | [Feedback](#) | [A-Z Index](#)

For more information on Java technology
and other software from Sun Microsystems, call:
(800) 786-7638
Outside the U.S. and Canada, dial your country's
[AT&T Direct Access Number](#) first.



Copyright © 1995-2001 [Sun Microsystems, Inc.](#)
All Rights Reserved. [Terms of Use](#). [Privacy Policy](#).



[Products & APIs](#)
[Developer Connection](#)
[Docs & Training](#)
[Online Support](#)
[Community Discussion](#)
[Industry News](#)
[Solutions Marketplace](#)
[Case Studies](#)

[Printable Page](#) 

■ Requires login

[Early Access](#) ■
[Downloads](#)

[Bug Database](#) ■
[Submit a Bug](#)
[View Database](#)

[Newsletters](#)
[Back Issues](#)
[Subscribe](#)

[Learning Centers](#)
[Articles](#)
[Bookshelf](#)
[Code Samples](#)
[New to Java](#)
[Question of the Week](#)
[Quizzes](#)
[Tech Tips](#)
[Tutorials](#)

[Forums](#)

Writing Advanced Applications

Chapter 4: Remote Method Invocation

[\[<<BACK\]](#) [\[CONTENTS\]](#) [\[NEXT>>\]](#)

The Remote Method Invocation (RMI) application programming interface (API) enables client and server communications over the net between programs written in the Java™ programming language. The Enterprise JavaBeans™ server transparently implements the necessary Remote Method Invocation (RMI) code so the client program can reference the Enterprise Beans running on the server and access them as if they are running locally to the client program.

Having RMI built into the Enterprise JavaBeans server is very convenient and saves you coding time, but if you need to use advanced RMI features or integrate RMI with an existing application, you need to override the default RMI implementation and write your own RMI code.

This chapter replaces the container-managed `RegistrationBean` from [Chapter 2: Entity and Session Beans](#) with an RMI-based registration server. The container-managed `SellerBean` from Chapter 2 is also changed to call the new RMI registration server using a Java 2 RMI `lookup` call.

[About RMI](#)

[RMI in the Auction Application](#)

- [Class Overview](#)
- [File Summary](#)
- [Compile the Example](#)
- [Start the RMI Registry](#)
- [Start the Remote Server](#)

[Establishing Remote Communications](#)

[RegistrationServer Class](#)

- [Exporting a Remote Object](#)
- [Passing by Value and by Reference](#)
- [Distributed Garbage Collection](#)

[Registration Interface](#)

About RMI

The RMI API lets you access a remote server object from a client program by making simple method calls on the server object. While other distributed architectures for accessing remote server objects such as Distributed Component Object Model (DCOM) and Common Object Request Broker Architecture (CORBA) return references to the remote object, the RMI API not only returns references, but provides these additional benefits.

The RMI API handles remote object references (call by reference) and can also return a copy of the object (call by value).

If the client program does not have local access to the class from which a local or remote object was instantiated, RMI services can download the class file.

Serialization and Data marshaling

To transfer objects, the RMI API uses the Serialization API to wrap (marshal) and unwrap (unmarshal) the objects. To marshal an object, the Serialization API converts the object to a stream of bytes, and to unmarshal an object, the Serialization API converts a stream of bytes into an object.

RMI over IIOP

One of the initial disadvantages to RMI was that its sole reliance on the Java platform to write the interfaces made integration into existing legacy systems difficult. However, RMI over Internet Inter-ORB Protocol (IIOP) discussed in [Chapter 4: Lookup Services](#) lets RMI communicate with any system or language that CORBA supports.

If you combine improved integration with the ability of RMI to work through firewalls using HTTP firewall proxying, you might find distributing your business logic using RMI is easier than a socket-based solution.

Note: Transferring code and data are key parts of the Jini™ System software specification. In fact, adding a

discovery and join service to the RMI services would create something very similar to what you get in the Jini architecture.

RMI in the Auction Application

The RMI-based [RegistrationServer](#) has the following new methods:

A new `create` method for creating a new user.

A new `find` method for finding a user.

A new `search` method for the custom search of users in the database.

The new custom search passes results back to the calling client by way of an RMI callback. The RMI callback custom search is similar to the `finder` methods used in the Bean- and container-managed examples from Chapters 2 and 3, except in the RMI version, it can take more time to generate the results because the remote registration server calls a remote method exported by the RMI-based [SellerBean](#) client.

If the calling client is written in the Java programming language, and is not, for example, a web page, the server can update the client as soon as the results are ready. But, the HTTP protocol used in most browsers does not allow results to be pushed to the client without a request for those results. This means the results web page is not created until the results are ready, which can add a small delay.

Class Overview

The two main classes in the RMI-based auction implementation are [SellerBean](#) and the remote [RegistrationServer](#). `SellerBean` is called from [AuctionServlet](#) to insert an auction item into the database, and check for low account balances.

The example models the Enterprise JavaBeans architecture in that a user's registration details are separate from the code to create and find the registration details. That is, the user's registration details provided by the [Registration.java](#) class are separate from the code to create and find a `Registration` object, which is in the [RegistrationHome.java](#) class.

The remote interface implementation in [RegistrationHome.java](#) is bound to the `rmiregistry`. When a client program wants to manipulate a user's registration details, it must first look up the

reference to the [RegistrationHome.java](#) object in the `rmiregistry`.

File Summary

All the source code files for the RMI-based example are described in the bullet list below.

[SellerBean.java](#): Client program that calls the `RegistrationServer.verifypasswd` and `RegistrationServer.findLowCreditAccounts` remote methods. `SellerBean` also exports its `updateResults` method that `RegistrationServer` calls when it completes its `RegistrationServer.findLowCreditAccounts` search.

[RegistrationServer.java](#): Remote server object that implements the `RegistrationHome` and `Registration` remote interfaces.

[Registration.java](#): Remote interface that declares the `getUser`, `verifypasswd`, and other remote methods for managing a user's registration details.

[RegistrationHome.java](#): remote interface that declares the `create`, `findByPrimaryKey`, and `findLowCreditAccounts` remote methods that create or return instances of registration details.

[RegistrationImpl.java](#): The `RegistrationServer.java` source file includes the implementation for the `Registration` remote interface as class `RegistrationImpl`

[RegistrationPK.java](#): Class that represents a user's registration details using just the primary key of the database record.

[ReturnResults.java](#): Remote interface that declares the `updateResults` method the `SellerBean` class implements as a callback.

[AuctionServlet.java](#): Modified version of the original `AuctionServlet` class where registration accounts are created by calling the RMI `RegistrationServer` directly. The auction servlet also calls the `SellerBean.auditAccounts` method, which returns a list of users with a low account balance.

The `auditAccounts` method is called with the following Uniform

Resource Locator (URL), which does a simple check to verify the request came from the local host.

```
http://phoenix.eng.sun.com:7001/
    AuctionServlet?action=auditAccounts
```

You also need a [java.policy](#) security policy file to grant the permissions needed to run the example on the Java 2 platforms.

Most RMI applications need the two socket permissions for socket and HTTP access to the specified ports. The two thread permissions, were listed in a stack trace as being needed for the `RegistrationImpl` class to create a new inner thread.

In the Java 2 platform, when a program does not have all the permissions it needs, the Java¹ virtual machine (VM) generates a stack trace that lists the permissions that need to be added to the security policy file. See [Chapter 9: Program Signing and Security](#) for more information.

```
grant {
    permission java.net.SocketPermission
        "*:1024-65535", "connect,accept,resolve";
    permission java.net.SocketPermission " *:80",
        "connect";
    permission java.lang.RuntimePermission
        "modifyThreadGroup";
    permission java.lang.RuntimePermission
        "modifyThread";
};
```

Compile the Example

Before describing the RMI-based code for the above classes, here is the command sequence to compile the example on the Unix and Win32 platforms:

Unix:

```
javac registration/Registration.java
javac registration/RegistrationPK.java
javac registration/RegistrationServer.java
javac registration/ReturnResults.java
javac seller/SellerBean.java
rmic -d . registration.RegistrationServer
rmic -d . registration.RegistrationImpl
rmic -d . seller.SellerBean
```

Win32:

```
javac registration\Registration.java
javac registration\RegistrationPK.java
javac registration\RegistrationServer.java
javac registration\ReturnResults.java
javac seller\SellerBean.java
```

```
rmic -d . registration.RegistrationServer
rmic -d . registration.RegistrationImpl
rmic -d . seller.SellerBean
```

Start the RMI Registry

Because you are using your own RMI code, you have to explicitly start the RMI Registry so the `SellerBean` object can find the remote Enterprise Beans. The `RegistrationServer` uses the RMI Registry to register or bind enterprise Beans that can be called remotely. The `SellerBean` client contacts the registry to look up and get references to the remote `AuctionItem` and `Registration` Enterprise Beans.

Because RMI allows code and data to be transferred, you must be sure the system classloader does not load extra classes that could be mistakenly sent to the client. In this example, extra classes would be the `Stub` and `Skel` class files, and the `RegistrationServer` and `RegistrationImpl` classes, and to prevent them being mistakenly sent, they should not appear anywhere in the `CLASSPATH` when you start the RMI Registry. Because the current path could be included automatically, you need to start the RMI Registry away from the code workspace too.

The following commands prevent the sending of extra classes by unsetting the `CLASSPATH` before starting the RMI Registry on the default 1099 port. You can specify a different port by adding the port number as follows: `rmiregistry 4321 &`. If you specify a different port number, you must specify the same port number in both you client `lookup` and server `rebind` calls.

Unix:
`export CLASSPATH=""`
`rmiregistry &`

Win32:
`unset CLASSPATH`
`start rmiregistry`

Start the Remote Server

Once the `rmiregistry` is running, you can start the remote server, `RegistrationServer`. The `RegistrationServer` program registers the name `registration2` with the `rmiregistry` name server, and any client can use this name to retrieve a reference to the remote server object, `RegistrationHome`.

To run the example, copy the `RegistrationServer` and `RegistrationImpl` classes and the associated stub classes a

remotely accessible area and start the server program.

Unix:

```
cp *_Stub.class
    /home/zelda/public_html/registration
cp RegistrationImpl.class
/home/zelda/public_html/registration
cd /home/zelda/public_html/registration
java -Djava.server.hostname=
    phoenix.eng.sun.com RegistrationServer
```

Windows:

```
copy *_Stub.class
    \home\zelda\public_html\registration
copy RegistrationImpl.class
    \home\zelda\public_html\registration
cd \home\zelda\public_html\registration
java -Djava.server.hostname=
    phoenix.eng.sun.com RegistrationServer
```

The following key properties used to configure RMI servers and clients. These properties can be set inside the program or supplied as command line properties to the Java VM.

The `java.rmi.server.codebase` property specifies where the publicly accessible classes are located. On the server this can be a simple file URL to point to the directory or JAR file that contains the classes. If the URL points to a directory, the URL must terminate with a file separator character, "/".

If you are not using a file URL, you will either need an HTTP server to download the remote classes or have to manually deliver the remote client stub and remote interface classes in, for example, a JAR file.

The `java.rmi.server.hostname` property is the complete host name of the server where the publicly accessible classes reside. This is only needed if the server has problems generating a fully qualified name by itself.

The `java.rmi.security.policy` property specifies the [policy file](#) with the permissions needed to run the remote server object and access the remote server classes for download.

Establishing Remote Communications

Client programs communicate with each other through the server. The server program consists of three files. The `Registration.java` and `RegistrationHome.java` remote interface files define the methods that can be called remotely, and the `RegistrationServer.java` class file defines the `RegistrationServer`

and `RegistrationImpl` classes that implement the methods.

To establish remote communications, both the client and server programs need to access the remote interface classes. The server needs the interface classes to generate the interface implementation, and the client uses the remote interface class to call the remote server method implementation.

For example, `SellerBean` creates a reference to `RegistrationHome`, the interface, and not `RegistrationServer`, the implementation, when it needs to create a user registration.

Besides the server interfaces and classes, you need stub and skeleton classes to establish remote communications. The stub and skeleton classes needed in this example are generated when you run the `rmic` compiler command on the `RegistrationServer` and `SellerBean` classes.

The generated `SellerBean`, `SellerBean_Stub.class` and `SellerBean_Skel.class` classes are needed for the callback from the server to the `SellerBean` client. It is the `_Stub.class` file on the client that marshals data to and unmarshals it from the server, while the `_Skel.class` class does the same for the server.

Note: In the Java 2 platform, the server side, `_Skel.class` file is no longer needed because its function has been taken over by the Java Virtual Machine classes

Data Marshaling

Marshaling and unmarshaling data means that when you call the `RegistrationHome.create` method from `SellerBean`, this call is forwarded to the `RegistrationServer_Stub.create` method. The `RegistrationServer_Stub.create` method wraps the method arguments and sends a serialized stream of bytes to the `RegistrationServer_Skel.create` method.



The `RegistrationServer_Skel.create` method unwraps the serialized bytestream, re-creates the arguments to the original `RegistrationHome.create` call, and returns the result of calling the real `RegistrationServer.create` method back along the same route, but this time wrapping the data on the

server side.

Marshaling and unmarshaling data is not without its complications. The first issue is serialized objects might be incompatible across Java Development Kit (JDK™) releases. A Serialized object has an identifier stored with the object that ties the serialized object to its release. If the RMI client and server complain about incompatible serial IDs, you might need to generate backward compatible stubs and skeletons using the `-vcompat` option to the `rmic` compiler.

Another issue is not all objects are serialized by default. The initial Bean-managed `RegistrationBean` object this example is based on returns an `Enumeration` object that contains `Registration` elements in a `Vector`. Returning this list from a remote method works fine, but when you try to send a vector as a parameter to a remote method, you get a runtime `Marshaling` exception in the Java 2 platform.

Fortunately, in the Java 2 platform the Collections API offers alternatives to previously unmarshable objects. In this example, an `ArrayList` from the Collections API replaces the `Vector`. If the Collections API is not an option, you can create a wrapper class that extends `Serializable` and provides `readObject` and `writeObject` method implementations to convert the object into a bytestream.

RegistrationServer Class

The [RegistrationServer](#) class extends

`java.rmi.server.UnicastRemoteObject` and implements the `create`, `findByPrimaryKey` and `findLowCreditAccounts` methods declared in the `RegistrationHome` interface. The [RegistrationServer.java](#) source file also includes the implementation for the `Registration` remote interface as class `RegistrationImpl`. `RegistrationImpl` also extends `UnicastRemoteObject`.

Exporting a Remote Object

Any object that you want to be remotely accessible needs to either extend `java.rmi.server.UnicastRemoteObject` or use the `exportObject` method from the `UnicastRemoteObject` class. If you extend `UnicastRemoteObject`, you also get the `equals`, `toString` and `hashCode` methods for the exported object.

Passing by Value and Passing by Reference

Although the `RegistrationImpl` class is not bound to the registry, it

is still referenced remotely because it is associated with the `RegistrationHome` return results. Because `RegistrationImpl` extends `UnicastRemoteObject`, its results are passed by reference, and so only one copy of that user's registration Bean exists in the Java VM at any one time.

In the case of reporting results such as in the `RegistrationServer.findLowCreditAccounts` method, the `RegistrationImpl` class copy of the remote object could be used instead. By simply not extending `UnicastRemoteObject` in the `RegistrationImpl` class definition, a new `Registration` object would be returned for each request. In effect the values were passed but not the reference to the object on the server.

Distributed Garbage Collection

Using remote references to objects on the server from a client outside the server's garbage collector introduces some potential problems with memory leaks. How does the server know it is holding onto a reference to a `Registration` object that is no longer being used by any clients because they aborted or a network connection was dropped?

To avoid potential memory leaks on the server from clients, RMI uses a leasing mechanism when giving out references to exported objects. When exporting an object, the Java VM increases the count for the number of references to this object and sets an expiration time, or lease time, for the new reference to this object.

When the lease expires, the reference count of this object is decreased and if it reaches 0, the object is set for garbage collection by the Java VM. It is up to the client that maintains this weak reference to the remote object to renew the lease if it needs the object beyond the lease time. A weak reference is a way to refer to an object in memory without keeping it from being garbage collected.

This lease time value is a configurable property measured in milliseconds. If you have a fast network, you could shorten the default value and create a large number of transient object references.

The following code sets the lease timeout to 2 minutes.

```
Property prop = System.getProperties();
prop.put("java.rmi.dgc.leaseValue", 120000);
```

The `create` and `findByPrimaryKey` methods are practically identical

to the other versions of the Registration Server. The main difference is that on the server side, the registration record is referenced as `RegistrationImpl`, which is the implementation of `Registration`. On the client side, `Registration` is used instead.

The `findLowCreditAccounts` method builds an `ArrayList` of serializable `RegistrationImpl` objects and calls a remote method in the `SellerBean` class to pass the results back. The results are generated by an inner `Thread` class so the method returns before the results are complete. The `SellerBean` object waits for the `updateAccounts` method to be called before displaying the HTML page. In a client written with the Java programming language, it would not need to wait, but could display the update in real time.

```
public class RegistrationServer
    extends UnicastRemoteObject
    implements RegistrationHome {

    public registration.RegistrationPK
        create(String theuser,
              String password,
              String emailaddress,
              String creditcard)
        throws registration.CreateException{
        // code to insert database record
    }

    public registration.Registration
        findByPrimaryKey(registration.RegistrationPK pk)
        throws registration.FinderException {
        if ((pk == null) || (pk.getUser() == null)) {
            throw new FinderException ();
        }
        return(refresh(pk));
    }

    private Registration refresh(RegistrationPK pk)
        throws FinderException {

        if(pk == null) {
            throw new FinderException ();
        }

        Connection con = null;
        PreparedStatement ps = null;
        try{
            con=getConnection();
            ps=con.prepareStatement("select password,
            emailaddress,
            creditcard,
            balance from registration where theuser = ?");
            ps.setString(1, pk.getUser());
            ps.executeQuery();
            ResultSet rs = ps.getResultSet();
            if(rs.next()) {
                RegistrationImpl reg=null;
                try{
```

```

        reg= new RegistrationImpl();
    }catch (RemoteException e) {}
        reg.theuser = pk.getUser();
        reg.password = rs.getString(1);
        reg.emailaddress = rs.getString(2);
        reg.creditcard = rs.getString(3);
        reg.balance = rs.getDouble(4);
        return reg;
    }else{
        throw new FinderException ();
    }
}catch (SQLException sqe) {
    throw new FinderException();
}finally {
    try{
        ps.close();
        con.close();
    }catch (Exception ignore) {}
}
}

public void findLowCreditAccounts(
    final ReturnResults client)
    throws FinderException {
    Runnable bgthread = new Runnable() {
    public void run() {
        Connection con = null;
        ResultSet rs = null;
        PreparedStatement ps = null;
        ArrayList ar = new ArrayList();

        try{
            con=getConnection();
            ps=con.prepareStatement("select theuser,
                balance from registration
                where balance < ?");
            ps.setDouble(1, 3.00);
            ps.executeQuery();
            rs = ps.getResultSet();
            RegistrationImpl reg=null;
            while (rs.next()) {
                try{
                    reg= new RegistrationImpl();
                }catch (RemoteException e) {}
                    reg.theuser = rs.getString(1);
                    reg.balance = rs.getDouble(2);
                    ar.add(reg);
                }
            }
            rs.close();
            client.updateResults(ar);
        }catch (Exception e) {
            System.out.println("findLowCreditAccounts: "+e);
            return;
        }
    }finally {
        try{
            if(rs != null) {
                rs.close();
            }
            if(ps != null) {

```

```

        ps.close();
    }
    if(con != null) {
        con.close();
    }
} catch (Exception ignore) {}
}
} //run
};
Thread t = new Thread(bgthread);
t.start();
}
}

```

The `main` method loads the JDBC™ pool driver. This version uses the Postgres database, installs the `RMISecurityManager`, and contacts the RMI registry to bind the `RegistrationHome` remote object to the name `registration2`. It does not need to bind the remote interface, `Registration` because that class is loaded when it is referenced by `RegistrationHome`.

By default, the server name uses port 1099. If you want to use a different port number, you can add it with a colon as follows: `kq6py:4321`. If you change the port here, you must start the [RMI Registry](#) with the same port number.

The `main` method also installs a `RMIFailureHandler`. If the server fails to create a server socket then the failure handler returns `true` which instructs the RMI server to retry the operation.

```

public static void main(String[] args){
    try {
        new pool.JDCCConnectionDriver(
            "postgresql.Driver",
            "jdbc:postgresql:ejbdemo",
            "postgres", "pass");
    } catch (Exception e){
        System.out.println(
            "error in loading JDBC driver");
        System.exit(1);
    }
    try {
        Properties env=System.getProperties();
        env.put("java.rmi.server.codebase",
            "http://phoenix.eng.sun.com/registration");
        RegistrationServer rs=
            new RegistrationServer();
        if (System.getSecurityManager() == null ) {
            System.setSecurityManager(
                new RMISecurityManager());
        }
        RMISocketFactory.setFailureHandler(
            new RMIFailureHandlerImpl());

        Naming.rebind("

```

```

        //phoenix.eng.sun.com/registration2",rs);
    }catch (Exception e) {
        System.out.println("Exception thrown "+e);
    }
}

class RMIFailureHandlerImpl
    implements RMIFailureHandler {
    public boolean failure(Exception ex ){
        System.out.println("exception "+ex+" caught");
        return true;
    }
}

```

Registration Interface

The [Registration](#) interface declares the methods implemented by `RegistrationImpl` in the `RegistrationServer.java` source file.

```

package registration;

import java.rmi.*;
import java.util.*;

public interface Registration extends Remote {
    boolean verifyPassword(String password)
        throws RemoteException;
    String getEmailAddress() throws RemoteException;
    String getUser() throws RemoteException;
    int adjustAccount(double amount)
        throws RemoteException;
    double getBalance() throws RemoteException;
}

```

RegistrationHome Interface

The [RegistrationHome](#) interface declares the methods implemented by the `RegistrationServer` class. These methods mirror the `Home` interface defined in the Enterprise JavaBeans example. The `findLowCreditAccounts` method takes a remote interface as its only parameter.

```

package registration;

import java.rmi.*;
import java.util.*;

public interface RegistrationHome extends Remote {
    RegistrationPK create(String theuser,
        String password,
        String emailaddress,
        String creditcard)
        throws CreateException,
        RemoteException;
}

```



```

        Registration findByPrimaryKey(RegistrationPK theuser)
            throws FinderException, RemoteException;

        public void findLowCreditAccounts(ReturnResults rr)
            throws FinderException, RemoteException;
    }

```

ReturnResults Interface

The [ReturnResults](#) interface declares the method implemented by the `SellerBean` class. The `updateResults` method is called from `RegistrationServer`.

```

package registration;

import java.rmi.*;
import java.util.*;

public interface ReturnResults extends Remote {
    public void updateResults(ArrayList results)
        throws FinderException, RemoteException;
}

```

SellerBean Class

The [SellerBean](#) class includes the callback method implementation and calls the `RegistrationServer` object using RMI. The `updateAccounts` method is made accessible by a call to `UnicastRemoteObject.exportObject(this);`. The `auditAccounts` method waits on a `Boolean` object.

The `updateAccounts` method sends a notify to all methods waiting on the `Boolean` object when it has been called from the server and receives the search results.

```

package seller;

import java.rmi.RemoteException;
import java.rmi.*;
import javax.ejb.*;
import java.util.*;
import java.text.NumberFormat;
import java.io.Serializable;
import javax.naming.*;
import auction.*;
import registration.*;
import java.rmi.server.UnicastRemoteObject;
import java.util.ArrayList;

public class SellerBean
    implements SessionBean, ReturnResults {

```

```

protected SessionContext ctx;
javax.naming.Context ectx;
Hashtable env = new Hashtable();
AuctionServlet callee=null;
Boolean ready=new Boolean("false");
ArrayList returned;

public int insertItem(String seller,
String password,
String description,
int auctiondays,
double startprice,
String summary)
throws RemoteException {

try{
RegistrationHome regRef = (
RegistrationHome)Naming.lookup(
"//phoenix.eng.sun.com/registration2");
RegistrationPK rpk= new RegistrationPK();
rpk.setUser(seller);
Registration newseller = (
Registration)regRef.findByPrimaryKey(rpk);
if((newseller == null) ||
(!newseller.verifyPassword(password))) {
return(Auction.INVALID_USER);
}

AuctionItemHome home = (
AuctionItemHome) ectx.lookup(
"auctionitems");
AuctionItem ai= home.create(seller,
description,
auctiondays,
startprice,
summary);
if(ai == null) {
return Auction.INVALID_ITEM;
}else{
return(ai.getId());
}
}catch(Exception e){
System.out.println("insert problem="+e);
return Auction.INVALID_ITEM;
}
}

public void updateResults(java.util.ArrayList ar)
throws RemoteException {
returned=ar;
synchronized(ready) {
ready.notifyAll();
}
}

public ArrayList auditAccounts() {
this.callee=callee;
try {
RegistrationHome regRef = (
RegistrationHome)Naming.lookup(

```

```

        "://phoenix.eng.sun.com/registration2");
        regRef.findLowCreditAccounts(this);
        synchronized(ready) {
            try {
                ready.wait();
            } catch (InterruptedException e){}
        }
        return (returned);
    } catch (Exception e) {
        System.out.println("error in creditAudit "+e);
    }
    return null;
}

public void ejbCreate()
    throws javax.ejb.CreateException,
           RemoteException {
    env.put(
        javax.naming.Context.INITIAL_CONTEXT_FACTORY,
        "weblogic.jndi.TengahInitialContextFactory");
    try{
        ctx = new InitialContext(env);
    } catch (NamingException e) {
        System.out.println(
            "problem contacting EJB server");
        throw new javax.ejb.CreateException();
    }
    Properties env=System.getProperties();
    env.put("java.rmi.server.codebase",
        "http://phoenix.eng.sun.com/registration");
    env.put("java.security.policy","java.policy");
    UnicastRemoteObject.exportObject(this);
}

public void setSessionContext(SessionContext ctx)
    throws RemoteException {
    this.ctx = ctx;
}

public void unsetSessionContext()
    throws RemoteException {
    ctx = null;
}


public void ejbRemove() {}
public void ejbActivate() throws RemoteException {
    System.out.println("activating seller bean");
}
public void ejbPassivate() throws RemoteException {
    System.out.println("passivating seller bean");
}
}

```

¹ As used on this web site,
the terms "Java virtual
machine" or "JVM" mean a virtual machine

for the Java platform.

[\[TOP\]](#)

Printable Page 

[This page was updated: 4-Jun-2001]

[Products & APIs](#) | [Developer Connection](#) | [Docs & Training](#) | [Online Support](#)
[Community Discussion](#) | [Industry News](#) | [Solutions Marketplace](#) | [Case Studies](#)

[Glossary](#) | [Feedback](#) | [A-Z Index](#)

For more information on Java technology
and other software from Sun Microsystems, call:
(800) 786-7638
Outside the U.S. and Canada, dial your country's
[AT&T Direct Access Number](#) first.



Copyright © 1995-2001 [Sun Microsystems, Inc.](#)
All Rights Reserved. [Terms of Use](#). [Privacy Policy](#).



[Products & APIs](#)
[Developer Connection](#)
[Docs & Training](#)
[Online Support](#)
[Community Discussion](#)
[Industry News](#)
[Solutions Marketplace](#)
[Case Studies](#)

Printable Page 

■ Requires login

Early Access ■
[Downloads](#)

Bug Database ■
[Submit a Bug](#)
[View Database](#)

Newsletters
[Back Issues](#)
[Subscribe](#)

Learning Centers
[Articles](#)
[Bookshelf](#)
[Code Samples](#)
[New to Java](#)
[Question of the Week](#)
[Quizzes](#)
[Tech Tips](#)
[Tutorials](#)

Forums

Writing Advanced Applications

Chapter 4: Common Object Request Broker Architecture (CORBA)

[\[<< BACK\]](#) [\[CONTENTS\]](#) [\[NEXT >>\]](#)

Both the Remote Method Invocation (RMI) and Enterprise JavaBeans™ auction application implementations use the Java™ language to implement the different auction service tiers. However, you might need to integrate with applications written in C, C++ or other languages and running on a myriad of operating systems and machines.

One way to integrate with other applications is to transmit data in a common format such as 8 bit characters over a TCP/IP socket. The disadvantage is you have to spend a fair amount of time deriving a messaging protocol and mapping the various data structures to and from the common transmission format so the data can be sent and received over the TCP/IP connection.

This is exactly where Common Object Request Broker Architecture (CORBA) and its Interface Definition Language (IDL) can help. IDL provides a common format to represent an object that can be distributed to other applications. The other applications might not even understand objects, but as long as they can provide a mapping between the common IDL format and their own data representations, the applications can share data.

This chapter describes the Java language to IDL mapping scheme, and how to replace the original container-managed `RegistrationBean` with its CORBA server equivalent. The `SellerBean.java` and `AuctionServlet.java` programs are changed to interoperate with the CORBA `RegistrationServer` program.

[IDL Mapping Scheme](#)

- [Quick Reference](#)
- [Setting up IDL Mappings](#)
- [Other IDL Types](#)

[CORBA in the Auction Application](#)

- [CORBA RegistrationServer](#)

Technology Centers

- [IDL Mappings File](#)
- [Compiling the IDL Mappings File](#)
- [Stub and Skeleton Files](#)

[Object Request Broker \(ORB\)](#)

- [Making the CORBA Server Accessible](#)
- [Plugging in a New ORB](#)
- [Naming Service Access by CORBA Clients](#)

[Helper and Holder Classes](#)

[Garbage Collection](#)

[CORBA Callbacks](#)

[Using the Any Type](#)

[Conclusion](#)

IDL Mapping Scheme

Many programming languages provide a mapping between their data types to the common denominator IDL format, and the Java language is no exception. The Java language can send objects defined by IDL to other CORBA distributed applications, and receive objects defined by IDL from other CORBA distributed applications.

This section describes the Java language to IDL mapping scheme and, where needed, presents issues you need to take into consideration.

Quick Reference

Here is a quick reference table of the Java language to CORBA IDL data types, and the runtime exceptions thrown when conversions fail. Data types in this table that need explanation are covered below.

Java Data Type	IDL Format	Runtime Exception
byte	octet	
boolean	boolean	
char	char	DATA_CONVERSION
char	wchar	
double	double	
float	float	
int	long	
int	unsigned long	

long	long long	
long	unsigned long long	
short	short	
short	unsigned short	
java.lang.String	string	DATA_CONVERSION
java.lang.String	wstring	MARSHAL

Unsigned Values: The Java data types `byte`, `short`, `int`, and `long` are represented by 8 bit, 16 bit, 32 bit and 64 bit two's-complement integers. This means a Java `short` value represents the range -2^{15} to $2^{15} - 1$ or -32768 to 32767 inclusive. The equivalent signed IDL type for a `short`, `short`, matches that range, but the unsigned IDL `short` type uses the range 0 to 2^{15} or 0 to 65535.

This means that in the case of a `short`, if an unsigned short value greater than 32767 is passed to a program written in the Java language, the `short` value is represented in the Java language as a negative number. This can cause confusion in boundary tests for a value greater than 32767 or less than 0.

IDL char Types: The Java language uses 16-bit unicode, but the IDL `char` and `string` types are 8-bit characters. You can map a Java `char` to an 8-bit IDL `char` to transmit multi-byte characters if you use an array to do it. However, the IDL wide char type `wchar` is specifically designed for languages with multi-byte characters and allocates a fixed number of bytes as needed to contain that language set for each and every letter.

When mapping between the Java language `char` type and the IDL `char` type, the `DATA_CONVERSION` exception is thrown if the character does not fit into 8 bits.

IDL string Types: The IDL `string` type can be thought of as a sequence of IDL `char` types, and also raises the `DATA_CONVERSION` exception. The IDL `wstring` type is equivalent to a sequence of `wchars` terminated by a `wchar NULL`.

An IDL `string` and `wstring` type can either have a fixed size or no maximum defined sized. If you try to map a `java.lang.String` to a fixed size or bounded IDL `string` and the `java.lang.String` is too large, a `MARSHAL` exception is raised.

Setting up IDL Mappings

Java language to IDL mappings are placed in a file with a `.idl` extension. The file is compiled so it can be accessed by CORBA programs that need to send and receive data. This section explains how to construct the mappings for package statements and the Java data types. The section below on [CORBA RegistrationServer Implementation](#) describes how to use this information to set up an IDL mapping file for the CORBA `Registration` server.

Java packages and interfaces: Java package statements are equivalent to the `module` type in IDL. The `module` types can be nested, which results in generated Java classes being created in nested sub-directories.

For example, if a CORBA program contains this package statement:

```
package registration;
```

the mappings file would have this IDL module mapping for it:

```
module registration {
};
```

If a CORBA program contains a package hierarchy like this

```
package registration.corba;
```

the equivalent IDL module mapping is this:

```
module registration {
  module corba {
  };
};
```

Distributed classes are defined as Java interfaces and map to the IDL interface type. IDL does not define access such as `public` or `private` like you find in the Java language, but does allow inheritance from other interfaces.

This example adds the Java `Registration` interface to an IDL `registration` module.

```
module registration {
  interface Registration {
  };
}
```

This example adds the Java `Registration` interface to an IDL `registration` module, and indicates the `Registration` interface inherits from the `User` interface.

```
module registration {
  interface Registration: User {
  };
}
```


Java methods: Java methods map to IDL operations. The IDL operation looks similar to a Java method except there is no concept of access control. You also have to help the IDL compiler by specifying which parameters are `in`, `inout` or `out`, defined as follows:

`in` - parameter is passed into the method but not changed.

`inout` - parameter is passed into the method and might be returned changed.

`out` - parameter might be returned changed.

This IDL mapping includes the `Registration` and `RegistrationHome` interface methods to IDL operations using one IDL module type.

```
module registration {

    interface Registration {
        boolean verifyPassword(in string password);
        string getEmailAddress();
        string getUser();
        long adjustAccount(in double amount);
        double getBalance();
    };

    interface RegistrationHome {
        Registration findByPrimaryKey(
            in RegistrationPK theuser)
            raises (FinderException);
    }
}
```

Java Arrays: Arrays in the Java language are mapped to the IDL `array` or IDL `sequence` type using a type definition.

This example maps the Java array `double balances[10]` to an IDL `array` type of the same size.

```
typedef double balances[10];
```

These examples map the Java array `double balances[10]` to an IDL `sequence` type. The first `typedef sequence` is an example of an unbounded sequence, and the second `typedef sequence` has the same size as the array.

```
typedef sequence<double> balances;
typedef sequence<double,10> balances;
```

Java Exception: Java exceptions are mapped to IDL exceptions. Operations use IDL exceptions by specifying them as a `raises` type.

This example maps the `CreateException` from the auction application to the IDL exception type, and adds the IDL `raises` type to the operation as follows. IDL exceptions follow C++ syntax, so instead of throwing an exception (as you would in the Java

language), the operation raises an exception.

```
exception CreateException {
};

interface RegistrationHome {
    RegistrationPK create(
        in string theuser,
        in string password,
        in string emailaddress,
        in string creditcard)
        raises (CreateException);
}
```

Other IDL types

These other basic IDL types do not have an exact equivalent in the Java language. Many of these should be familiar if you have used C or C++. The Java language provides a mapping for these types so a program written in the Java language can receive data from programs written in C or C++.

- IDL attribute
- IDL enum
- IDL struct
- IDL union
- IDL Any
- IDL Principal
- IDL Object

IDL attribute: The IDL `attribute` type is similar to the `get` and `set` methods used to access fields in the JavaBeans™ software.

In the case of a value declared as an IDL attribute, the IDL compiler generates two methods of the same name as the IDL attribute. One method returns the field and the other method sets it. For example, this `attribute` type:

```
interface RegistrationPK {
    attribute string theuser;
};
```

defines these methods

```
//return user
String theuser();
//set user
void theuser(String arg);
```

IDL enum: The Java language has an `Enumeration` class for representing a collection of data. The IDL `enum` type is different because it is declared as a data type and not a data collection.

The IDL `enum` type is a list of values that can be referenced by name instead of by their position in the list. In the example, you can see that referring to an IDL `enum` status code by name is more readable than referring to it by its number. This line maps `static final int` values in the `final class` `LoginError`. You can reference the values as you would reference a static field:

```
LoginError.INVALID_USER.
```

```
enum LoginError {
    INVALID_USER, WRONG_PASSWORD, TIMEOUT};
```

Here is a version of the `enum` type that includes a preceding underscore that can be used in `switch` statements:

```
switch (problem) {
    case LoginError._INVALID_USER:
        System.out.println("please login again");
        break;
}
```

IDL struct: An IDL `struct` type can be compared to a Java class that has only fields, which is how it is mapped by the IDL compiler.

This example declares an IDL `struct`. Note that IDL types can reference other IDL types. In this example `LoginError` is from the `enum` type declared above.

```
struct ErrorHandler {
    LoginError errortype;
    short retries;
};
```

IDL union: An IDL `union` can represent one type from a list of types defined for that union. The IDL `union` maps to a Java class of the same name with a `discriminator` method used for determining the type of this union.

This example maps the `GlobalErrors` union to a Java class by the name of `GlobalErrors`. A default case `case: DEFAULT` could be added to handle any elements that might be in the `LoginErrors` `enum` type, and not specified with a `case` statement here.

```
union GlobalErrors switch (LoginErrors) {
    case: INVALID_USER: string message;
    case: WRONG_PASSWORD: long attempts;
    case: TIMEOUT: long timeout;
};
```

In a program written in the Java language, the `GlobalErrors` union class is created as follows:

```
GlobalErrors ge = new GlobalErrors();
ge.message("please login again");
```

The `INVALID_USER` value is retrieved like this:

```

switch (ge.discriminator().value()) {
    case: LoginError._INVALID_USER:
        System.out.println(ge.message);
        break;
}

```

Any type: If you do not know what type is going to be passed or returned to an operation, you can use the `Any` type mapping, which can represent any IDL type. The following operation returns and passes an unknown type:

```

interface RegistrationHome {
    Any customSearch(Any searchField, out count);
};

```

To first create a type of `Any`, request the type from the Object Request Broker (ORB). To set a value in a type of `Any`, use an `insert_<type>` method. To retrieve a value, use the `extract_<type>` method.

This example requests an object of type `Any`, and uses the `insert_type` method to set a value.

```

Any sfield = orb.create_any();
sfield.insert_long(34);

```

The `Any` type has an assigned `TypeCode` value that you can query using `type().kind().value()` on the object. The following example shows a test for the `TypeCode` `double`. This example includes a reference to the IDL `TypeCode` find out which type the `Any` object contains. The `TypeCode` is used for all objects. You can analyze the type of a CORBA object using the `_type()` or `type()` methods as shown here.

```

public Any customSearch(Any searchField, IntHolder count){
    if(searchField.type().kind().value() == TCKind._tk_double){
        // return number of balances greater than supplied amount
        double findBalance=searchField.extract_double();
    }
}

```

Principal: The `Principal` type identifies the owner of a CORBA object, for example, a user name. The value can be interrogated from the `request_principal` field of the CORBA `RequestHeader` class to make the identification. More comprehensive security and authorization is available in the CORBA security service. **Object:** The `Object` type is a CORBA object. If you need to send Java objects, you have to either translate them into an IDL type or use a mechanism to serialize them when they are transferred.

CORBA in the Auction Application

The container-managed [RegistrationBean](#) from the auction application is completely replaced with a standalone CORBA

[RegistrationServer](#) that implements the registration service. The `CORBA RegistrationServer` is built by creating and compiling an IDL mappings file so client programs can communicate with the registration server.

The `SellerBean.java` and `AuctionServlet.java` files are updated to look up the CORBA registration server.

CORBA RegistrationServer Implementation

This section describes the [Registration.idl](#) file, which maps the `RegistrationHome` and `Registration` remote interfaces from the Enterprise JavaBean auction application to their IDL equivalents and shows how to compile the `Registration.idl` file into CORBA registration server classes.

The CORBA registration server implements the `create` and `findByPrimaryKey` methods from the original `RegistrationBean.java` file, and is enhanced with the following two new methods to help illustrate CORBA callbacks and how to use the `Any` type.

`findLowCreditAccounts(in ReturnResults rr)`, which uses a [callback](#) to return a list of accounts with a low balance.

`any customSearch(in any searchfield, out long count)`, which returns a different search result depending on the search [field type](#) submitted.

IDL Mappings File

Here is the [Registration.idl](#) file that maps the data types and methods used in the `RegistrationHome` and `Registration` programs to their IDL equivalents.

```
module registration {

interface Registration {
    boolean verifyPassword(in string password);
    string getEmailAddress();
    string getUser();
    long adjustAccount(in double amount);
    double getBalance();
};

interface RegistrationPK {
    attribute string theuser;
};

enum LoginError {INVALIDUSER, WRONGPASSWORD, TIMEOUT};
```

```

exception CreateException {
};

exception FinderException {
};

typedef sequence<Registration> IDLArrayList;

interface ReturnResults {
    void updateResults(in IDLArrayList results)
        raises (FinderException);
};

interface RegistrationHome {
    RegistrationPK create(in string theuser,
        in string password,
        in string emailaddress,
        in string creditcard)
        raises (CreateException);

    Registration findByPrimaryKey(
        in RegistrationPK theuser)
        raises (FinderException);
    void findLowCreditAccounts(in ReturnResults rr)
        raises (FinderException);
    any customSearch(in any searchfield, out long count);
};
};

```

Compiling the IDL Mappings File

The IDL file has to be converted into Java classes that can be used in the CORBA distributed network. The Java 2 platform compiles .idl files using the program `idltojava`. This program will be eventually replaced with the `idlj` command.

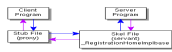
The `-fno-cpp` arguments indicate there is no C++ compiler installed.

```
idltojava -fno-cpp Registration.idl
```

Other Java IDL compilers should also work, for example, `jidl` from ORBacus can generate classes that can be used by the Java 2 ORB.

Stubs and Skeletons

Corba and RMI are similar in that compilation generates a stub file for the client and a skeleton file for the server. The stub (or proxy), and skeleton (or servant) are used to marshal and unmarshal data between the client and the server. The skeleton (or servant) is implemented by the server. In this example, the IDL `RegistrationHome` interface mapping generates a `_RegistrationHomeImplBase` class (the skeleton or servant class) that the generated `RegistrationServer` class extends.



When requesting a remote CORBA object or calling a remote method, the client call passes through the stub class before reaching the server. This proxy class invokes CORBA requests for the client program. The following example is the code automatically generated for the `RegistrationHomeStub.java` class.

```

org.omg.CORBA.Request r = _request("create");
r.set_return_type(
    registration.RegistrationPKHelper.type());
org.omg.CORBA.Any _theuser = r.add_in_arg();
  
```

Object Request Broker

The center of the CORBA distributed network is the Object Request Broker or ORB. The ORB is involved in marshaling and unmarshaling objects between the client and server. Other services such as the Naming Service and Event Service work with the ORB.

The Java 2 platform includes an ORB in the distribution called the IDL ORB. This ORB is different from many other ORBs because it does not include a distinct Basic Object Adapter (BOA) or Portable Object Adapter (POA).

An object adapter manages the creation and lifecycle of objects in the CORBA distributed space. This can be compared to the container in the Enterprise JavaBeans server managing the lifecycle of the session and entity beans.

The [AuctionServlet](#) and [SellerBean](#) programs create and initialize a Java 2 ORB like this:

```
ORB orb = ORB.init(args, null);
```

In the [RegistrationServer](#) program, the server object to be distributed is bound to the ORB using the `connect` method:

```
RegistrationServer rs = new RegistrationServer();
orb.connect(rs);
```

An object connected to an ORB can be removed with the `disconnect` method:

```
orb.disconnect(rs);
```

Once connected to a CORBA server object, the Java 2 ORB keeps the server alive and waits for client requests to the CORBA server.

```
java.lang.Object sync = new java.lang.Object();
synchronized(sync) {
    sync.wait();
}
```

Making the CORBA Server Accessible

Although this object is now being managed by the ORB, the clients do not yet have a mechanism to find the remote object. This can be solved by binding the CORBA server object to a naming service.

The Java 2 naming service is called `tnameserv`. The naming service by default uses port 900; however, this value can be changed by setting the argument `-ORBInitialPort portnumber` when starting `tnameserv` or setting the property `org.omg.CORBA.ORBInitialPort` when starting the client and server processes.

These next sections describes the `main` method from the `RegistrationServer` class.

```
java.util.Properties props=System.getProperties();
props.put("org.omg.CORBA.ORBInitialPort", "1050");
System.setProperties(props);
ORB orb = ORB.init(args, props);
```

The next lines show the initial naming reference is initialized by requesting the service called `NameService`. The `NamingContext` is retrieved and the name built up and bound to the naming service as `NameComponent` elements. The name in this example has a root called `auction` with this object being bound as `RegistrationBean` from that `auction` root. The name could be compared to a class by the name of `auction.RegistrationBean`.

```
org.omg.CORBA.Object nameServiceObj =
    orb.resolve_initial_references("NameService") ;
NamingContext nctx =
    NamingContextHelper.narrow(nameServiceObj);
NameComponent[] fullname = new NameComponent[2];
fullname[0] = new NameComponent("auction", "");
fullname[1] = new NameComponent(
    "RegistrationBean", "");

NameComponent[] tempComponent = new NameComponent[1];
for(int i=0; i < fullname.length-1; i++ ) {
    tempComponent[0]= fullname[i];
    try{
        nctx=nctx.bind_new_context(tempComponent);
    }catch (Exception e){
        System.out.println("bind new"+e);}
}
```



```

    }
    tempComponent[0]=fullname[fullname.length-1];
    try{
        nctx.rebind(tempComponent, rs);
    }catch (Exception e){
        System.out.println("rebind"+e);
    }
}

```

Plugging in a new ORB

The Java 2 IDL ORB does not currently include some of the services available in many other commercial ORBS such as the security or event (notification) services. You can use another ORB in the Java 2 runtime by configuring two properties and including any necessary object adapter code.

Using a new ORB in the registration server requires the `org.omg.CORBA.ORBClass` and `org.omg.CORBA.ORBSingletonClass` properties point to the appropriate ORB classes. In this example the ORBacus ORB is used instead of the Java 2 IDL ORB. To use another ORB, the code below should be plugged into the `RegistrationServer.main` method.

In the example code, a `SingletonClass` ORB is used. The `SingletonClass` ORB is not a full ORB, and is primarily used as a factory for `TypeCodes`. The call to `ORB.init()` in the last line creates the `Singleton` ORB.

```

Properties props= System.getProperties();
props.put("org.omg.CORBA.ORBClass",
        "com.ooc.CORBA.ORB");
props.put("org.omg.CORBA.ORBSingletonClass",
        "com.ooc.CORBA.ORBSingleton");
System.setProperties(props);
ORB orb = ORB.init(args, props) ;

```

In the Java 2 IDL, there is no distinct object adapter. As shown in the example code segment below, using the Basic Object Adapter from ORBacus requires an explicit cast to the ORBacus ORB. The Broker Object Architecture (BOA) is notified that the object is ready to be distributed by calling the `impl_is_ready(null)` method.

```

BOA boa = ((com.ooc.CORBA.ORB)orb).BOA_init(
        args, props);
...
boa.impl_is_ready(null);

```

Although both the `ORBSingletonClass` and `ORBClass` ORBs build the object name using `NameComponent`, you have to use a different ORBacus Naming Service. The `CosNaming.Server` service is started as follows where the `-OAhost` parameter is optional:

```
java com.ooc.CosNaming.Server -OAhost localhost -OAport 1060
```

Once the naming service is started, the server and client programs find the naming service using the IIOP protocol to the host and port named when starting the Naming service:

```
java registration.RegistrationServer
    -ORBservice NameService
    iiop://localhost:1060/DefaultNamingContext
```

Naming Service Access by CORBA Clients

CORBA clients access the naming service in a similar way to the server, except that instead of binding a name, the client resolves the name built from the `NameComponents`.

The `AuctionServlet` and `SellerBean` classes use the following code to look up the CORBA server:

```
NameComponent[] fullname = new NameComponent[2];
fullname[0] = new NameComponent("auction", "");
fullname[1] = new NameComponent(
    "RegistrationBean", "");

RegistrationHome regRef =
    RegistrationHomeHelper.narrow(
        nctx.resolve(fullname));
```

In the case of the ORBacus ORB, the clients also need a Basic Object Adapter if callbacks are used as in the `SellerBean.auditAccounts` method. The naming context helper is also configured differently for the ORBacus server started earlier:

```
Object obj =
    ((com.ooc.CORBA.ORB)orb).get_inet_object (
        "localhost",
        1060,
        "DefaultNamingContext");
NamingContext nctx = NamingContextHelper.narrow(obj);
```

Helper and Holder classes

References to remote objects in CORBA use a `Helper` class to retrieve a value from that object. A commonly used method is the `Helper narrow` method, which ensures the object is cast correctly.

`Holder` classes hold values returned when using `inout` or `out` parameters in a method. The caller first instantiates the appropriate `Holder` class for that type and retrieves the value from the class when the call returns. In the next example, the count value for `customSearch` is set and retrieved after `customSearch` has been called. On the server side the count value is set by calling `count.value=newvalue`.

```

IntHolder count= new IntHolder();
sfield=regRef.customSearch(sfield,count);
System.out.println("count now set to "+count.value);

```

Garbage Collection

Unlike RMI, CORBA does not have a distributed garbage collection mechanism. References to an object are local to the client proxy and the server servant. This means each Java¹ virtual machine (JVM) is free to reclaim that object and garbage collect it if there are no longer references to it. If an object is no longer needed on the server, the `orb.disconnect(object)` needs to be called to allow the object to be garbage collected.

CORBA Callbacks

The new `findLowCreditAccounts` method is called from the `AuctionServlet` using the Uniform Resource Locator (URL) `http://localhost:7001/AuctionServlet?action=auditAccounts`.

The `AuctionServlet.auditAccounts` method calls the `SellerBean.auditAccounts` method, which returns an `ArrayList` of **Registration records**.

```

//AuctionServlet.java
private void auditAccounts(ServletOutputStream out,
    HttpServletRequest request) throws IOException{

//    ...

    SellerHome home = (SellerHome) ctx.lookup("seller");
    Seller si= home.create();

    if(si != null) {
        ArrayList ar=si.auditAccounts();
        for(Iterator i=ar.iterator(); i.hasNext(); ) {
            Registration user=(Registration)(i.next());
            addLine("<TD>"+user.getUser() +
                "<TD><TD>"+user.getBalance()+
                "<TD><TR>", out);
        }
        addLine("<TABLE>", out);
    }
}

```

The `SellerBean` object calls the CORBA

`RegistrationHome.findLowCreditAccounts` method implemented in the `RegistrationServer.java` file, and passes a reference to itself. The reference is passed as the `SellerBean` class implements the `ReturnResults` interface declared in the `Registration.idl`.

```

//SellerBean.java
public ArrayList auditAccounts() {
    try{

```

```

NameComponent[] fullname = new NameComponent[2];
fullname[0] = new NameComponent("auction", "");
fullname[1] = new NameComponent(
    "RegistrationBean", "");

RegistrationHome regRef =
    RegistrationHomeHelper.narrow(
        nctx.resolve(fullname));
regRef.findLowCreditAccounts(this);
synchronized(ready) {
    try{
        ready.wait();
    }catch (InterruptedException e){}
}
return (returned);
}catch (Exception e) {
    System.out.println("error in auditAccounts "+e);
}
return null;
}

```

The `RegistrationServer.findLowCreditAccounts` **method** **retrieves** **user** **records** **from** **the** **database** **registration** **table** **that** **have** **a** **credit** **value** **less** **than** **three**. **It** **then** **returns** **the** **list** **of** **Registration** **records** **in** **an** **ArrayList** **by** **calling** **the** `SellerBean.updateResults` **method** **that** **it** **has** **a** **reference** **to**.

```

//RegistrationServer.java
public void findLowCreditAccounts(
    final ReturnResults client)
    throws Finder Exception {
    Runnable bgthread = new Runnable() {
    public void run() {
        Connection con = null;
        ResultSet rs = null;
        PreparedStatement ps = null;
        ArrayList ar = new ArrayList();

        try{
            con=getConnection();
            ps=con.prepareStatement(
                "select theuser,
                balance from registration
                where balance < ?");
            ps.setDouble(1, 3.00);
            ps.executeQuery();
            rs = ps.getResultSet();
            RegistrationImpl reg=null;
            while (rs.next()) {
                try{
                    reg= new RegistrationImpl();
                }catch (Exception e) {
                    System.out.println("creating reg"+e);
                }
                reg.theuser = rs.getString(1);
                reg.balance = rs.getDouble(2);
                ar.add(reg);
            }
            rs.close();
        }
    }
}

```

```

        RegistrationImpl[] regarray =
        (RegistrationImpl [])ar.toArray(
        new RegistrationImpl[0]);
        client.updateResults(regarray);
    }catch (Exception e) {
        System.out.println(
            "findLowCreditAccounts: "+e);
        return;
    }
    finally {
    try{
        if(rs != null) {
            rs.close();
        }
        if(ps != null) {
            ps.close();
        }
        if(con != null) {
            con.close();
        }
    }catch (Exception ignore) {}
    }
    }//run
    };
    Thread t = new Thread(bgthread);
    t.start();
    }

```

The SellerBean.updateResults method updates the global ArrayList of Registration records returned by the RegistrationServer object and notifies the SellerBean/auditAccounts method that it can return that ArrayList of Registration records to the AuctionServlet.

```

public void updateResults(Registration[] ar)
    throws registration.FinderException {
    if(ar == null) {
        throw new registration.FinderException();
    }
    try{
        for(int i=0; i< ar.length; i++) {
            returned.add(ar[i]);
        }
    }catch (Exception e) {
        System.out.println("updateResults="+e);
        throw new registration.FinderException();
    }
    synchronized(ready) {
        ready.notifyAll();
    }
}

```

Using the Any type

The RegistrationServer.customSearch method uses the IDL Any type to pass in and return results. The customSearch is called by the AuctionServlet as follows:

```

http://localhost.eng.sun.com:7001/
AuctionServlet?action=customSearch&searchfield=2

```

The `searchfield` parameter can be set to a number or a string. The `AuctionServlet.customFind` method passes the search field directly to the `SellerBean.customFind` method and retrieves a `String` that is then displayed to the user.

```

private void customSearch(ServletOutputStream out,
                          HttpServletRequest request)
    throws IOException{

    String text = "Custom Search";
    String searchField=request.getParameter(
        "searchfield");

    setTitle(out, "Custom Search");
    if(searchField == null ) {
        addLine("Error: SearchField was empty", out);
        out.flush();
        return;
    }
    try{
        addLine("<BR>"+text, out);
        SellerHome home = (SellerHome)
            ctx.lookup("seller");
        Seller si= home.create();
        if(si != null) {
            String displayMessage=si.customFind(
                searchField);
            if(displayMessage != null ) {
                addLine(displayMessage+"<BR>", out);
            }
        }
    }catch (Exception e) {
        addLine("AuctionServlet customFind error",out);
        System.out.println("AuctionServlet " +
            "<customFind>:"+e);
    }
    out.flush();
}

```

The `SellerBean.customFind` method calls the `RegistrationHome` object implemented in the `RegistrationServer.java` class, and depending on whether the `searchField` can be converted into a double or a string, inserts this value into an object of type `Any`. The `Any` object is created by a call to the ORB, `orb.create_any()`;

The `customFind` method also uses an `out` parameter, `count`, of type `int` that returns the number of records found. The value of `count` is retrieved using `count.value` when the call returns.

```

//SellerBean.java
public String customFind(String searchField)
    throws javax.ejb.FinderException,
        RemoteException{

    int total=-1;
    IntHolder count= new IntHolder();

```

```

try{
    NameComponent[] fullname = new NameComponent[2];
    fullname[0] = new NameComponent("auction", "");
    fullname[1] = new NameComponent(
        "RegistrationBean", "");

    RegistrationHome regRef =
        RegistrationHomeHelper.narrow(
            nctx.resolve(fullname));
    if(regRef == null ) {
        System.out.println(
            "cannot contact RegistrationHome");
        throw new javax.ejb.FinderException();
    }
    Any sfield=orb.create_any();
    Double balance;
    try{
        balance=Double.valueOf(searchField);
        try {
            sfield.insert_double(balance.doubleValue());
        }catch (Exception e) {
            return("Problem with search value"+balance);
        }
        sfield=regRef.customSearch(sfield,count);
        if(sfield != null ) {
            total=sfield.extract_long();
        }
        return(total+"
accounts are below optimal level from" +
count.value+" records");
    }catch (NumberFormatException e) {
        sfield.insert_string(searchField);
        Registration reg;
        if((reg=RegistrationHelper.extract(
            regRef.customSearch(
                sfield,count)))
            != null ) {
            return("Found user "+reg.getUser() +"
who has email address "+
reg.getEmailAddress());
        }else {
            return("No users found who have email address " +
searchField);
        }
    }
}catch(Exception e){
    System.out.println("customFind problem="+e);
    throw new javax.ejb.FinderException();
}
}
}

```

The return value from the call to `customFind` is extracted into an object of type `Any` and a `String` is constructed with the output displayed to the user. For simple types, the `extract_<type>` method of the `Any` object can be used. However, for the `Registration` type, the `RegistrationHelper` class is used.

```
Registration reg =
```

```

RegistrationHelper.extract(
    regRef.customSearch(sfield,count))

```

The RegistrationServer.customSearch method determines the type of Object being passed in the searchField parameter by checking the .type().kind().value() of the Any object.

```

if(searchField.type().kind().value() ==
    TCKind._tk_double)

```

Finally, because the customSearch method returns an object of type Any, a call to orb.create_any() is required. For simple types like double, the insert_<type> method is used. For a Registration record, the RegistrationHelper class is used:

```

RegistrationHelper.insert(returnResults, regarray[0]).

```

```

//RegistrationServer.java
public Any customSearch(Any searchField,
    IntHolder count){
    Any returnResults= orb.create_any();

    int tmpcount=count.value;
    if(searchField.type().kind().value() ==
        TCKind._tk_double){
// return number of balances greater
// than supplied amount
        double findBalance=searchField.extract_double();
        Connection con = null;
        ResultSet rs = null;
        PreparedStatement ps = null;
        try{
            con=getConnection();
            ps=con.prepareStatement("select count(*) from
                registration where balance < ?");
            ps.setDouble(1, findBalance);
            ps.executeQuery();
            rs = ps.getResultSet();
            if(rs.next()) {
                tmpcount = rs.getInt(1);
            }
            count.value=tmpcount;
            rs.close();
        }catch (Exception e) {
            System.out.println("custom search: "+e);
            returnResults.insert_long(-1);
            return(returnResults);
        }
    finally {
        try{
            if(rs != null) { rs.close(); }
            if(ps != null) { ps.close(); }
            if(con != null) { con.close(); }
        } catch (Exception ignore) {}
    }
    returnResults.insert_long(tmpcount);
    return(returnResults);
}
else if(searchField.type().kind().value() ==
    TCKind._tk_string) {
    // return email addresses that match supplied address
    String findEmail=searchField.extract_string();

```



```

Connection con = null;
ResultSet rs = null;
PreparedStatement ps = null;
ArrayList ar = new ArrayList();
RegistrationImpl reg=null;
try{
    con=getConnection();
    ps=con.prepareStatement("select theuser,
        emailaddress from registration
        where emailaddress like ?");
    ps.setString(1, findEmail);
    ps.executeQuery();
    rs = ps.getResultSet();
    while (rs.next()) {
        reg= new RegistrationImpl();
        reg.theuser = rs.getString(1);
        reg.emailaddress = rs.getString(2);
        ar.add(reg);
    }
    rs.close();


    RegistrationImpl[] regarray =
        (RegistrationImpl [])ar.toArray(
            new RegistrationImpl[0]);
    RegistrationHelper.insert(
                returnResults,
                regarray[0]);
    return(returnResults);
}catch (Exception e) {
    System.out.println("custom search: "+e);
    return(returnResults);
}
finally {
    try{
        if(rs != null) { rs.close(); }
        if(ps != null) { ps.close(); }
        if(con != null) { con.close(); }
    } catch (Exception ignore) {}
}
}
return(returnResults);
}

```

Conclusion

As you can see, converting the application to use RMI or CORBA requires very little change to core programs. The main difference has been the initialization and naming service. By abstracting these two areas in your application away from the business logic you ease migration between different distributed object architectures.

¹ As used on this web site, the terms "Java virtual machine" or "JVM" mean a virtual machine for the Java platform.

Printable Page 

[This page was updated: 4-Jun-2001]

[Products & APIs](#) | [Developer Connection](#) | [Docs & Training](#) | [Online Support](#)
[Community Discussion](#) | [Industry News](#) | [Solutions Marketplace](#) | [Case Studies](#)

[Glossary](#) | [Feedback](#) | [A-Z Index](#)

For more information on Java technology
and other software from Sun Microsystems, call:
(800) 786-7638
Outside the U.S. and Canada, dial your country's
[AT&T Direct Access Number](#) first.



Copyright © 1995-2001 [Sun Microsystems, Inc.](#)
All Rights Reserved. [Terms of Use](#). [Privacy Policy](#).



[Products & APIs](#)
[Developer Connection](#)
[Docs & Training](#)
[Online Support](#)
[Community Discussion](#)
[Industry News](#)
[Solutions Marketplace](#)
[Case Studies](#)

[Training Index](#)

Writing Advanced Applications

Chapter 4: JDBC™ Technology

[\[<<BACK\]](#)
[\[CONTENTS\]](#)
[\[NEXT>>\]](#)

[Printable Page](#) 

■ Requires login

[Early Access](#) ■
[Downloads](#)

[Bug Database](#) ■
[Submit a Bug](#)
[View Database](#)

[Newsletters](#)
[Back Issues](#)
[Subscribe](#)

[Learning Centers](#)
[Articles](#)
[Bookshelf](#)
[Code Samples](#)
[New to Java](#)
[Question of the Week](#)
[Quizzes](#)
[Tech Tips](#)
[Tutorials](#)

[Forums](#)

The Bean-managed Enterprise JavaBeans™ auction application with its Remote Method Invocation (RMI) and Common Object Request Broker (CORBA) variants have used simple JDBC™ calls to retrieve and update information from a database using a JDBC connection pool. By default, JDBC database access involves opening a database connection, running SQL commands in a statement, processing the returned results, and closing the database connection.

Overall, the default approach works well for low volume database access, but how do you manage a large number of requests that update many related tables at once and still ensure data integrity? This section explains how with the following topics.

[JDBC Drivers](#)

[Database Connections](#)

[Statements](#)

- [Callable Statements](#)
- [Statements](#)
- [Prepared Statements](#)

[Caching Database Results](#)

[Result Sets](#)

[Scrolling Result Sets](#)

[Controlling Transactions](#)

[Escaping Characters](#)

[Mapping Database Types](#)

[Mapping Date types](#)

JDBC Drivers

The connection to the database is handled by the JDBC Driver

Technology Centers class. The Java™ SDK contains only one JDBC driver, a `jdbc-odbc` bridge that can communicate with an existing Open DataBase Conectivity (ODBC) driver. Other databases need a JDBC driver specific to that database.

To get a general idea of what the JDBC driver does, you can examine the `JDBCConnectionDriver.java` file. The `JDBCConnectionDriver` class implements the `java.sql.Driver` class and acts as a pass-through driver by forwarding JDBC requests to the real database JDBC Driver. The JDBC driver class is loaded with a call to `Class.forName(drivename)`.

These next lines of code show how to load three different JDBC driver classes:

```
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
Class.forName("postgresql.Driver");
Class.forName("oracle.jdbc.driver.OracleDriver");
```

Each JDBC driver is configured to understand a specific URL so multiple JDBC drivers can be loaded at any one time. When you specify a URL at connect time, the first matching JDBC driver is selected.

The `jdbc-odbc` bridge accepts Uniform Resource Locators (URLs) starting with `jdbc:odbc:` and uses the next field in that URL to specify the data source name. The data source name identifies the particular database scheme you wish to access. The URL can also include more details on how to contact the database and enter the account.

```
//access the ejbdemo tables
String url = "jdbc:odbc:ejbdemo";
```

This next example contains the Oracle SQL*net information on the particular database called `ejbdemo` on machine `dbmachine`

```
String url = "jdbc:oracle:thin:user/password@(
    description=(address_list=(
        address=(protocol=tcp)
        (host=dbmachine)(port=1521)))(source_route=yes)
    (connect_data=(sid=ejbdemo)))";
```

This next examples uses `mysql` to connect to the `ejbdemo` database on the local machine. The login user name and password details are also included.

```
String url =
    "jdbc:mysql://localhost/ejbdemo?user=user;
    password=pass";
```

JDBC drivers are divided into four types. Drivers may also be categorized as pure Java or thin drivers to indicate if they are used for client applications (pure Java drivers) or applets (thin drivers). Newer drivers are usually Type 3 or 4. The four types are as

follows:

Type 1 Drivers

Type 1 JDBC drivers are the bridge drivers such as the jdbc-odbc bridge. These drivers rely on an intermediary such as ODBC to transfer the SQL calls to the database. Bridge drivers often rely on native code, although the jdbc-odbc library native code is part of the Java¹ 2 virtual machine.

Type 2 Drivers

Type 2 Drivers use the existing database API to communicate with the database on the client. Although Type 2 drivers are faster than Type 1 drivers, Type 2 drivers use native code and require additional permissions to work in an applet.

A Type 2 driver might need client-side database code to connect over the network.

Type 3 Drivers

Type 3 Drivers call the database API on the server. JDBC requests from the client are first proxied to the JDBC Driver on the server to run. Type 3 and 4 drivers can be used by thin clients as they need no native code.

Type 4 Drivers

The highest level of driver reimplements the database network API in the Java language. Type 4 drivers can also be used on thin clients as they also have no native code.

Database Connections

A database connection can be established with a call to the `DriverManager.getConnection` method. The call takes a URL that identifies the database, and optionally, the database login user name and password.

```
Connection con = DriverManager.getConnection(url);
Connection con = DriverManager.getConnection(url,
                                           "user", "password");
```

After a connection is established, a statement can be run against the database. The results of the statement can be retrieved and the connection closed.

One useful feature of the `DriverManager` class is the `setLogStream`

method. You can use this method to generate tracing information so help you diagnose connection problems that would normally not be visible. To generate tracing information, just call the method like this:

```
DriverManager.setLogStream(System.out);
```

The [Connection Pooling](#) section in Chapter 8 shows you how to improve the throughput of JDBC connections by not closing the connection once the statement completes. Each JDBC connection to a database incurs overhead in opening a new socket and using the username and password to log into the database. Reusing the connections reduces the overhead. The Connection Pool keeps a list of open connections and clears any connections that cannot be reused.

Statements

There are three basic types of SQL statements used in the JDBC API: `CallableStatement`, `Statement`, and `PreparedStatement`. When a `Statement` or `PreparedStatement` is sent to the database, the database driver translates it into a format the underlying database can recognize.

Callable Statements

Once you have established a connection to a database, you can use the `Connection.prepareCall` method to create a callable statement. A callable statement lets you execute SQL stored procedures.

This next example creates a `CallableStatement` object with three parameters for storing account login information.

```
CallableStatement cs =
    con.prepareCall("{call accountlogin(?,?,?)}");
cs.setString(1,theuser);
cs.setString(2,password);
cs.registerOutParameter(3,Types.DATE);

cs.executeQuery();
Date lastLogin = cs.getDate(3);
```

Statements

The `Statement` interface lets you execute a simple SQL statement with no parameters. The SQL instructions are inserted into the `Statement` object when the `Statement.executeXXX` method is called.

Query Statement: This code segment creates a `Statement` object and calls the `Statement.executeQuery` method to select text from the `dba` database. The results of the query are returned in a `ResultSet` object. How to retrieve results from a `ResultSet` object is explained in [Result Sets](#) below.

```
Statement stmt = con.createStatement();
ResultSet results = stmt.executeQuery(
    "SELECT TEXT FROM dba ");
```

Update Statement: This code segment creates a `Statement` object and calls the `Statement.executeUpdate` method to add an email address to a table in the `dba` database.

```
String updateString =
    "INSERT INTO dba VALUES (some text)";
int count = stmt.executeUpdate(updateString);
```

Prepared Statements

The `PreparedStatement` interface descends from the `Statement` interface and uses a template to create a SQL request. Use a `PreparedStatement` to send precompiled SQL statements with one or more parameters.

Query PreparedStatement: You create a `PreparedStatement` object by specifying the template definition and parameter placeholders. The parameter data is inserted into the `PreparedStatement` object by calling its `setXXX` methods and specifying the parameter and its data. The SQL instructions and parameters are sent to the database when the `executeXXX` method is called.

This code segment creates a `PreparedStatement` object to select user data based on the user's email address. The question mark ("?") indicates this statement has one parameter.

```
PreparedStatement pstmt = con.prepareStatement(
    select theuser from
    registration where
    emailAddress like ?");
//Initialize first parameter with email address
pstmt.setString(1, emailAddress);
ResultSet results = ps.executeQuery();
```

Once the `PreparedStatement` template is initialized, only the changed values are inserted for each call.

```
pstmt.setString(1, anotherEmailAddress);
```

Note: Not all database drivers compile prepared statements.

Update PreparedStatement: This code segment creates a `PreparedStatement` object to update a seller's registration record. The template has five parameters, which are set with five calls to the appropriate `PreparedStatement.setXXX` methods.

```
PreparedStatement ps = con.prepareStatement(
    "insert into registration(theuser, password,
        emailaddress, creditcard,
        balance) values (
            ?, ?, ?, ?, ?)");
ps.setString(1, theuser);
ps.setString(2, password);
ps.setString(3, emailaddress);
ps.setString(4, creditcard);
ps.setDouble(5, balance);
ps.executeUpdate();
```

Caching Database results

The `PreparedStatement` concept of reusing requests can be extended to caching the results of a JDBC call. For example, an auction item description remains the same until the seller changes it. If the item receives thousands of requests, the results of the statement: `query "select description from auctionitems where item_id='4000343'"` might be stored more efficiently in a hash table.

Storing results in a hash table requires the JDBC call be intercepted before creating a real statement to return the cached results, and the cache entry be cleared if there is a corresponding update to that `item_id`.

Result Sets

The `ResultSet` interface manages access to data returned from a query. The data returned equals one row in a database table. Some queries return one row of data while many queries return multiple rows of data.

You use `getType` methods to retrieve data from specific columns for each row returned by the query. This example retrieves the `TEXT` column from all tables with a `TEXT` column in the `dba` database. The `results.next` method moves to the next retrieved row until all returned rows are processed.

```
Statement stmt = con.createStatement();
ResultSet results = stmt.executeQuery(
```



```

                                "SELECT TEXT FROM dba ");
while(results.next()){
    String s = results.getString("TEXT");
    displayText.append(s + "\n");
}
stmt.close();

```

Scrolling Result Sets

Before JDBC 2.0, JDBC drivers returned read-only result sets with cursors that moved in one direction, forwards. Each element was retrieved by calling the `next` method on the result set.

JDBC 2.0 introduces scrollable results sets whose values can be read and updated if reading and updating is supported by the underlying database. With scrollable result sets, any row can be selected at random, and the result set can be traversed forwards and backwards.

One advantage to the new result set is you can update a set of matching rows without having to issue an additional `executeUpdate` call. The updates are made using JDBC calls and so no custom SQL commands need to be generated. This improves the portability of the database code you create.

Both `Statements` and `PreparedStatement` have an additional constructor that accepts a scroll type and an update type parameter. The scroll type value can be one of the following values:

`ResultSet.TYPE_FORWARD_ONLY`

Default behavior in JDBC 1.0, application can only call `next()` on the result set.

`ResultSet.SCROLL_SENSITIVE`

`ResultSet` is fully navigable and updates are reflected in the result set as they occur.

`ResultSet.SCROLL_INSENSITIVE`

Result set is fully navigable, but updates are only visible after the result set is closed. You need to create a new result set to see the results.

The update type parameter can be one of the following two values:

`ResultSet.CONCUR_READ_ONLY`

The result set is read only.

`ResultSet.CONCUR_UPDATABLE`

The result set can be updated.

You can verify that your database supports these types by calling `con.getMetaData().supportsResultSetConcurrency()` method as shown here.

```

Connection con = getConnection();
if(con.getMetaData().supportsResultSetConcurrency(
    ResultSet.SCROLL_INSENSITIVE,
    ResultSet.CONCUR_UPDATABLE)) {

    PreparedStatement pstmt = con.prepareStatement(
        "select password, emailaddress,
        creditcard, balance from
        registration where theuser = ?",
        ResultSet.SCROLL_INSENSITIVE,
        ResultSet.CONCUR_UPDATABLE);
}

```

Navigating the ResultSet

The fully scrollable result set returns a cursor which can be moved using simple commands. By default the result set cursor points to the row before the first row of the result set. A call to `next()` retrieves the first result set row. The cursor can also be moved by calling one of the following `ResultSet` methods:

`beforeFirst()`: Default position. Puts cursor before the first row of the result set.

`first()`: Puts cursor on the first row of the result set.

`last()`: Puts cursor before the last row of the result set.

`afterLast()` Puts cursor beyond last row of the result set. Calls to `previous` moves backwards through the `ResultSet`.

`absolute(pos)`: Puts cursor at the row number position where `absolute(1)` is the first row and `absolute(-1)` is the last row.

`relative(pos)`: Puts cursor at a row relative to its current position where `relative(1)` moves row cursor one row forward.

Updating the Result Set

You can update a value in a result set by calling the `ResultSet.update<type>` method on the row where the cursor is positioned. The type value here is the same used when retrieving a value from the result set, for example, `updateString` updates a `String` value in the result set.

This next code updates the balance for a user from the result set

created earlier. The update applies only to the result set until the call to `rs.updateRow()`, which updates the underlying database. Closing the result set before calling `updateRow` will lose any edits applied to the result set.

```
rs.first();
updateDouble("balance",
    rs.getDouble("balance") - 5.00);
```

Inserting a new row uses the same `update<type>` methods. The only difference being that the method `rs.moveToInsertRow` is called before and `rs.insertRow()` is called after the fields have been initialized. You can delete the current row with a call to `rs.deleteRow()`.

Batch Jobs

By default, every JDBC statement is sent to the database individually. Apart from the additional network requests, this process incurs additional delays if a transaction spans several of the statements. JDBC 2.0 lets you submit multiple statements at one time with the `addBatch` method.

This next code shows how to use the `addBatch` statement. The calls to `stmt.addBatch` append statements to the original `Statement`, and the call to `executeBatch` submits the entire statement with all the appends to the database.

```
Statement stmt = con.createStatement();
stmt.addBatch(
    "update registration set balance=balance-5.00
    where theuser="+theuser);
stmt.addBatch(
    "insert into auctionitems(
        description, startprice)
    values("+description+", "+startprice+)");

int[] results = stmt.executeBatch();
```

The return result of the `addBatch` method is an array of row counts affected for each statement executed in the batch job. If a problem occurred, a `java.sql.BatchUpdateException` is thrown. An incomplete array of row counts can be obtained from `BatchUpdateException` by calling its `getUpdateCounts` method.

Storing Classes, Images and Other Large Objects

Many databases can store binary data as part of a row if the database field is assigned a `long raw`, `longvarbinary`, or other similar type. These fields can accommodate up to two Gigabytes of data. This means if you can convert the data into a binary stream

or array of bytes, it can be stored and retrieved from the database in the same way you would store a string or double.

This technique can be used to store and retrieve images and Java objects.

Storing and retrieving an image: It is very easy to store an object that can be serialized or converted to a byte array. Unfortunately, `java.awt.Image` is not `Serializable`. However, as shown in this next code example, you can store the image data to a file and store the the information in the file as bytes in a database binary field.

```
int itemnumber=400456;

File file = new File(itemnumber+".jpg");
FileInputStream fis = new FileInputStream(file);
PreparedStatement pstmt = con.prepareStatement(
    "update auctionitems
    set theimage=? where id= ?");
pstmt.setBinaryStream(1, fis, (int)file.length());
pstmt.setInt(2, itemnumber);
pstmt.executeUpdate();
pstmt.close();
fis.close();
```

To retrieve this image and create a byte array that can be passed to `createImage`, do the following:

```
int itemnumber=400456;
byte[] imageBytes;

PreparedStatement pstmt = con.prepareStatement(
    "select theimage from auctionitems where id= ?");
pstmt.setInt(1, itemnumber);
ResultSet rs=pstmt.executeQuery();
if(rs.next()) {
    imageBytes = rs.getBytes(1);
}
pstmt.close();
rs.close();

Image auctionimage =
    Toolkit.getDefaultToolkit().createImage(
        imageBytes);
```

Storing and retrieving an object: A class can be serialized to a binary database field in much the same way as the image was in the previous example. In this example, the `RegistrationImpl` class is changed to support default serialization by adding `implements Serializable` to the Class declaration.

Next, a `ByteArrayInputStream` is created to be passed as the JDBC Binary Stream. To create the `ByteArrayInputStream`, `RegistrationImpl` is first piped through an `ObjectOutputStream` to an

underlying `ByteArrayInputStream` with a call to `RegistrationImpl.writeObject`. **The** `ByteArrayInputStream` **is then converted to a byte array, which can then be used to create the** `ByteArrayInputStream`. **The** `create` **method in** `RegistrationServer.java` **is changed as follows:**

```

public registration.RegistrationPK create(
    String theuser,
    String password,
    String emailaddress,
    String creditcard)
    throws registration.CreateException{

    double balance=0;
    Connection con = null;
    PreparedStatement ps = null;;

    try {
        con=getConnection();
        RegistrationImpl reg= new RegistrationImpl();
        reg.theuser = theuser;
        reg.password = password;
        reg.emailaddress = emailaddress;
        reg.creditcard = creditcard;
        reg.balance = balance;

        ByteArrayOutputStream regStore =
            new ByteArrayOutputStream();
        ObjectOutputStream regObjectStream =
            new ObjectOutputStream(regStore);
            regObjectStream.writeObject(reg);

        byte[] regBytes=regStore.toByteArray();
        regObjectStream.close();
        regStore.close();
        ByteArrayInputStream regArrayStream =
            new ByteArrayInputStream(regBytes);
        ps=con.prepareStatement(
            "insert into registration (
            theuser, theclass) values (?, ?)");
        ps.setString(1, theuser);
        ps.setBinaryStream(2, regArrayStream,
            regBytes.length);

        if (ps.executeUpdate() != 1) {
            throw new CreateException ();
        }
        RegistrationPK primaryKey =
            new RegistrationPKImpl();
        primaryKey.theuser(theuser);
        return primaryKey;
    } catch (IOException ioe) {
        throw new CreateException ();
    } catch (CreateException ce) {
        throw ce;
    } catch (SQLException sqe) {
        System.out.println("sqe="+sqe);
        throw new CreateException ();
    } finally {

```

```

        try {
            ps.close();
            con.close();
        } catch (Exception ignore) {
        }
    }
}

```

The object is retrieved and reconstructed by extracting the bytes from the database, creating a `ByteArrayInputStream` from those bytes to be read from an `ObjectInputStream`, and calling `readObject` to create the instance again.

This next example shows the changes needed to the `RegistrationServer.refresh` method to retrieve the registration instance from the database.

```

private Registration refresh(RegistrationPK pk)
    throws FinderException {

    if (pk == null) {
        throw new FinderException ();
    }
    Connection con = null;
    PreparedStatement ps = null;
    try {
        con=getConnection();
        ps=con.prepareStatement("
select theclass from
        registration where theuser = ?");
        ps.setString(1, pk.theuser());
        ps.executeQuery();
        ResultSet rs = ps.getResultSet();
        if(rs.next()){
            byte[] regBytes = rs.getBytes(1);
            ByteArrayInputStream regArrayStream =
                new ByteArrayInputStream(regBytes);
            ObjectInputStream regObjectStream =
                new ObjectInputStream(
                    regArrayStream);
            RegistrationImpl reg=
                (RegistrationImpl)
                    regObjectStream.readObject();
            return reg;
        }
        else {
            throw new FinderException ();
        }
    } catch (Exception sqe) {
        System.out.println("exception "+sqe);
        throw new FinderException ();
    }
    finally {
        try {
            rs.close();
            ps.close();
            con.close();
        }
        catch (Exception ignore) {}
    }
}

```

```
    }
}
```

BLOBs and CLOBs: Storing large fields in a table with the other data is not necessarily the optimum place especially if the data has a variable size. One way to handle large, variable sized objects is with the Large Objects (LOBs) type. LOBs use a locator, essentially a pointer, in the database record that points to the real database field.

There are two types of LOBs: Binary Large Objects (BLOBs) and Character Large Objects (CLOBs). When you access a BLOB or CLOB, the data is not copied to the client. To retrieve the actual data from a result set, you have to retrieve the pointer with a call to `BLOB blob=getBlob(1)` or `CLOB clob=getClob(1)`, and then retrieve the data with a call to `blob.getBinaryStream()` or `clob.getBinaryStream()`.

Controlling Transactions

By default, JDBC statements are processed in full auto-commit mode. This mode works well for a single database query, but if an operation depends on several database statements that all have to complete successfully or the entire operation is cancelled, a finer transaction is needed.

A description of transaction isolation levels is covered in more detail in [Chapter 3: Data and Transaction Management](#). To use transaction management in the JDBC platform, you first need to disable the full auto-commit mode by calling:

```
Connection con= getConnection();
con.setAutoCommit(false);
```

At this point, you can either `commit` any following JDBC statements or `undo` any updates by calling the `Connection.rollback` method. The `rollback` call is commonly placed in the Exception handler, although it can be placed anywhere in the transaction flow.

This next example inserts an auction item and decrements the user's balance. If the balance is less than zero, the entire transaction is rolled back and the auction item is removed.

```
public int insertItem(String seller,
    String password,
    String description,
    int auctiondays,
    double startprice,
    String summary) {
    Connection con = null;
    int count=0;
```

```

double balance=0;
java.sql.Date enddate, startdate;
Statement stmt=null;

PreparedStatement ps = null;
try {
    con=getConnection();
    con.setAutoCommit(false);
    stmt= con.createStatement();
    stmt.executeQuery(
        "select counter from auctionitems");
    ResultSet rs = stmt.getResultSet();
    if(rs.next()) {
        count=rs.getInt(1);
    }
    Calendar currenttime=Calendar.getInstance();
    java.util.Date currentdate=currenttime.getTime();
    startdate=new java.sql.Date(
        currentdate.getTime());
    currenttime.add(Calendar.DATE, auctiondays);
    enddate=new java.sql.Date((
        currenttime.getTime()).getTime());

    ps=con.prepareStatement(
        "insert into auctionitems(
            id, description, startdate, enddate,
            startprice, summary)
            values (?, ?, ?, ?, ?, ?)");
    ps.setInt(1, count);
    ps.setString(2, description);
    ps.setDate(3, startdate);
    ps.setDate(4, enddate);
    ps.setDouble(5, startprice);
    ps.setString(6, summary);
    ps.executeUpdate();
    ps.close();

    ps=con.prepareStatement(
        "update registration
        set balance=balance -0.50
        where theuser= ?");
    ps.setString(1, seller);
    ps.close();
    stmt= con.createStatement();
    stmt.executeQuery(
        "select balance from registration
        where theuser='"+seller+"'");
    rs = stmt.getResultSet();
    if(rs.next()) {
        balance=rs.getDouble(1);
    }
    stmt.close();
    if(balance <0) {
        con.rollback();
        con.close();
        return (-1);
    }
}

stmt= con.createStatement();
stmt.executeUpdate(

```



```

        "update auctionitems set
        counter=counter+1");
stmt.close();
con.commit();
con.close();
return(0);
} catch(SQLException e) {
    try {
        con.rollback();
        con.close();
        stmt.close();
        ps.close();
    }catch (Exception ignore){}
}
return (0);
}

```

Escaping Characters

The JDBC API provides the `escape` keyword so you can specify the character you want to use to escape characters. For example, if you want to use the percent sign (%) as the percent sign and not have it interpreted as the SQL wildcard used in SQL `LIKE` queries, you have to escape it with the escape character you specify with the `escape` keyword.

This next statements shows how you would use the `escape` keyword to look for the value 10%.

```

stmt.executeQuery(
    "select tax from sales where tax like
    '10\%' {escape '\\}");

```

If your program stores names and addresses to the database entered from the command line or by way of a user interface, the single quotes (') symbol might appear in the data. Passing single quotes directly into a SQL string causes problems when the SQL statement is parsed because SQL gives this symbol another meaning unless it is escaped.

To solve this problem, the following method escapes any ' symbol found in the input line. This method can be extended to escape any other characters such as commas , that the database or database driver might interpret another way.

```

static public String escapeLine(String s) {
    String retvalue = s;
    if (s.indexOf ("'") != -1 ) {
        StringBuffer hold = new StringBuffer();
        char c;
        for(int i=0; i < s.length(); i++ ) {
            if ((c=s.charAt(i)) == '\'' ) {
                hold.append ("''");
            }else {

```

```

        hold.append(c);
    }
}
retvalue = hold.toString();
}
return retvalue;
}

```

However, if you use a `PreparedStatement` instead of a simple `Statement`, most of these escape problems go away. For example, instead of this line with the escape sequence:

```

stmt.executeQuery(
"select tax from sales where tax like
    '10\%' {escape '\'}");

```

You could use this line:

```

preparedstmt = C.prepareStatement(
    "update tax set tax = ?");

```

Mapping Database Types

Apart from a few JDBC types such as `INTEGER` that are represented as an `INTEGER` in most popular databases, you might find that the JDBC type for a table column does not match the type as it is represented in the database. This means calls to `ResultSet.getObject`, `PreparedStatement.setObject` and `CallableStatement.getObject()` will very likely fail.

Your program can determine the database column type from the database meta data and use that information to check the value before retrieving it. This next code checks that the value is in fact type `INTEGER` before retrieving its value.

```

int count=0;
Connection con=getConnection();
Statement stmt= con.createStatement();
stmt.executeQuery(
    "select counter from auctionitems");
ResultSet rs = stmt.getResultSet();
if(rs.next()) {
    if(rs.getMetaData().getColumnType(1) ==
        Types.INTEGER) {
        Integer i=(Integer)rs.getObject(1);
        count=i.intValue();
    }
}
rs.close();

```

Mapping Date types

The `DATE` type is where most mismatches occur. This is because the `java.util.Date` class represents both Date and Time, but SQL has the following three types to represent data and time information:

A `DATE` type that represents the date only (03/23/99).

A `TIME` type that specifies the time only (12:03:59)

A `TIMESTAMP` that represents time value in nanoseconds.

These three additional types are provided in the `java.sql` package as `java.sql.Date`, `java.sql.Time` and `java.sql.Timestamp` and are all subclasses of `java.util.Date`. This means you can use convert `java.util.Date` values to the type you need to be compatible with the database type.

Note: The `Timestamp` class loses precision when it is converted to a `java.util.Date` because `java.util.Date` does not contain a nanosecond field, it is better to not convert a `Timestamp` instance if the value will be written back to the database.

This example uses the `java.sql.Date` class to convert the `java.util.Date` value returned by the call to `Calendar.getTime` to a `java.sql.Date`.


```
Calendar currenttime=Calendar.getInstance();
java.sql.Date startdate=
    new java.sql.Date((
        currenttime.getTime()).getTime());
```

You can also use the `java.text.SimpleDateFormat` class to do the conversion. This example uses the `java.text.SimpleDateFormat` class to convert a `java.util.Date` object to a `java.sql.Date` object:

```
SimpleDateFormat template =
    new SimpleDateFormat("yyyy-MM-dd");
java.util.Date enddate =
    new java.util.Date("10/31/99");
java.sql.Date sqlDate =
    java.sql.Date.valueOf(
        template.format(enddate));
```

If you find a database date representation cannot be mapped to a Java type with a call to `getObject` or `getDate`, retrieve the value with a call to `getString` and format the string as a `Date` value using the `SimpleDateFormat` class shown above.

¹ As used on this web site, the terms "Java virtual machine" or "JVM" mean a virtual machine for the Java platform.

Printable Page 

[This page was updated: 4-Jun-2001]

[Products & APIs](#) | [Developer Connection](#) | [Docs & Training](#) | [Online Support](#)
[Community Discussion](#) | [Industry News](#) | [Solutions Marketplace](#) | [Case Studies](#)

[Glossary](#) | [Feedback](#) | [A-Z Index](#)

For more information on Java technology
and other software from Sun Microsystems, call:
(800) 786-7638
Outside the U.S. and Canada, dial your country's
[AT&T Direct Access Number](#) first.



Copyright © 1995-2001 [Sun Microsystems, Inc.](#)
All Rights Reserved. [Terms of Use](#). [Privacy Policy](#).



[Products & APIs](#)
[Developer Connection](#)
[Docs & Training](#)
[Online Support](#)
[Community Discussion](#)
[Industry News](#)
[Solutions Marketplace](#)
[Case Studies](#)

[Training Index](#)

Writing Advanced Applications

Chapter 4 Continued: Servlets

[\[<<BACK\]](#) [\[CONTENTS\]](#) [\[NEXT>>\]](#)

[Printable Page](#) 

■ Requires login

[Early Access](#) ■
[Downloads](#)

[Bug Database](#) ■
[Submit a Bug](#)
[View Database](#)

[Newsletters](#)
[Back Issues](#)
[Subscribe](#)

[Learning Centers](#)
[Articles](#)
[Bookshelf](#)
[Code Samples](#)
[New to Java](#)
[Question of the Week](#)
[Quizzes](#)
[Tech Tips](#)
[Tutorials](#)

[Forums](#)

A servlet is a server-side program written in the Java™ programming language that interacts with clients and is usually tied to a HyperText Transfer Protocol (HTTP) server. One common use for a servlet is to extend a web server by providing dynamic web content.

Servlets have an advantage over other technologies in that they are compiled, have threading capability built in, and provide a secure programming environment. Even web sites that previously did not provide servlet support, can do so now by using programs such as JRun or the Java module for the Apache web server.

The web-based [auction application](#) uses a servlet to accept and process buyer and seller input through the browser and dynamically return auction item information to the browser. The AuctionServlet program is created by extending the HttpServlet class. The HttpServlet class provides a framework for handling HTTP requests and responses.

This section examines the AuctionServlet and includes information on how to use Cookie and Session objects in a servlet.

[HttpServlet](#)

[The init Method](#)

[The destroy Method](#)

[The service Method](#)

[HTTP Requests](#)

[Using Cookies in Servlets](#)

- [Setting a Cookie](#)
- [Retrieving a Cookie](#)
- [Generating Sessions](#)
- [Preventing Page Caching](#)

- [Restricting Access and Redirection](#)
 - [HTTP Error Codes](#)
 - [Reading GET and POST Values](#)
 - [Threading](#)
 - [HTTPS](#)
-

HttpServlet

The [AuctionServlet](#) class extends `HttpServlet`, which is an abstract class.

```
public class AuctionServlet extends HttpServlet {
```

A servlet can be either loaded when the web server starts up or loaded when requested by way of an HTTP URL that specifies the servlet. The servlet is usually loaded by a separate classloader in the web server because this allows the servlet to be reloaded by unloading the class loader that loaded the servlet class. However, if the servlet depends on other classes and one of those classes changes, you will need to update the date stamp on the servlet for it to reload.

After a servlet loads, the first stage in its lifecycle is the web server calls the servlet's `init` method. Once loaded and initialized, the next stage in the servlet's lifecycle is to serve requests. The servlet serves requests through its `service`, `doGet`, or `doPost` method implementations.

The servlet can optionally implement a `destroy` method to perform clean-up operations before the web server unloads the servlet.

The `init` Method

The `init` method is only called once by the web server when the servlet is first started. The `init` method is passed a `ServletConfig` object containing initialization information pertaining to the web server where the application is running.

The `ServletConfig` object is used to access information maintained by the web server including values from the `initArgs` parameter in the servlet properties file. Code in the `init` method uses the `ServletConfig` object to retrieve the `initArgs` values by calling the `config.getInitParameter("parameter")` method.

The `AuctionServlet.init` method also contacts the Enterprise JavaBeans server to create a context (`ctx`) object. The `ctx` object is

used in the `service` method to establish a connection with the Enterprise JavaBeans server.

```
Context ctx=null;
private String detailsTemplate;

public void init(ServletConfig config)
    throws ServletException{
    super.init(config);
    try {
        ctx = getInitialContext();
    }catch (Exception e){
        System.err.println(
            "failed to contact EJB server"+e);
    }
    try {
        detailsTemplate=readFile(
            config.getInitParameter("detailstemplate"));
    } catch(IOException e) {
        System.err.println(
            "Error in AuctionServlet <init>"+e);
    }
}
```

The destroy Method

The `destroy` method is a lifecycle method implemented by servlets that need to save their state between servlet loading and unloading. For example, the `destroy` method would save the current servlet state, and the next time the servlet is loaded, that saved state would be retrieved by the `init` method. You should be aware that the `destroy` method might not be called if the server machine crashes.

```
public void destroy() {
    saveServletState();
}
```

The service Method

The `AuctionServlet` is an HTTP servlet that handles client requests and generates responses through its `service` method. It accepts as parameters the `HttpServletRequest` and `HttpServletResponse` request and response objects.

`HttpServletRequest` contains the headers and input streams sent from the client to the server.

`HttpServletResponse` is the output stream that is used to send information from the servlet back to the client.

The `service` method handles standard HTTP client requests received by way of its `HttpServletRequest` parameter by delegating

the request to one of the following methods designed to handle that request. The different types of requests are described in the [HTTP Requests](#) section.

doGet for GET, conditional GET, and HEAD requests.

doPost for POST requests.

doPut for PUT requests.

doDelete for DELETE requests.

doOptions for OPTIONS requests.

doTrace for TRACE requests.

The `AuctionServlet` program provides its own `service` method implementation that calls one of the following methods based on the value returned by the call to

`cmd=request.getParameter("action")`. These method implementations match the default implementations provided in the `doGet` and `doPost` methods called by the default `service` method, but add some auction application-specific functionality for looking up Enterprise Beans.

`listAllItems(out)`

`listAllNewItems(out)`

`listClosingItems(out)`

`insertItem(out, request)`

`itemDetails(out, request)`

`itemBid(out, request)`

`registerUser(out, request)`

```
public void service(HttpServletRequest request,
                   HttpServletResponse response)
    throws IOException {

    String cmd;
    response.setContentType("text/html");
    ServletOutputStream out = response.getOutputStream();
    if (ctx == null ) {
        try {
            ctx = getInitialContext();
        }catch (Exception e){
            System.err.println(
                "failed to contact EJB server"+e);
        }
    }

    cmd=request.getParameter("action");
    if(cmd !=null) {
        if(cmd.equals("list")) {
            listAllItems(out);
        }else
        if(cmd.equals("newlist")) {
            listAllNewItems(out);
        }else if(cmd.equals("search")) {
            searchItems(out, request);
        }else if(cmd.equals("close")) {
            listClosingItems(out);
        }else if(cmd.equals("insert")) {
```



```

        insertItem(out, request);
    }else if (cmd.equals("details")) {
        itemDetails(out, request );
    }else if (cmd.equals("bid")) {
        itemBid(out, request) ;
    }else if (cmd.equals("register")) {
        registerUser(out, request);
    }
}else{
    // no command set
    setTitle(out, "error");
}
setFooter(out);
out.flush();
}

```

HTTP Requests

A request is a message sent from a client program such as a browser to a server program. The first line of the request message contains a method that indicates the action to perform on the incoming Uniform Resource Locator (URL). The two commonly used mechanisms for sending information to the server are `POST` and `GET`.

`GET` requests might pass parameters to a URL by appending them to the URL. `GET` requests can be bookmarked and emailed and include the information to the URL of the response.

`POST` requests might pass additional data to a URL by directly sending it to the server separately from the URL. `POST` requests cannot be bookmarked or emailed and do not change the URL of the response.

`PUT` requests are the reverse of `GET` requests. Instead of reading the page, `PUT` requests write (or store) the page.

`DELETE` requests are for removing web pages.

`OPTIONS` requests are for getting information about the communication options available on the request/response chain.

`TRACE` requests are for testing or diagnostic purposes because they let the client see what is being received at the other end of the request chain.

Using Cookies in servlets

`HTTP` cookies are essentially custom `HTTP` headers that are passed between a client and a server. Although cookies are not

overwhelmingly popular, they do enable state to be shared between the two machines. For example, when a user logs into a site, a cookie can maintain a reference verifying the user has passed the password check and can use that reference to identify that same user on future visits.

Cookies are normally associated with a server. If you set the domain to `.java.sun.com`, then the cookie is associated with the domain. If no domain is set, the cookie is only associated with the server that created the cookie.

Setting a Cookie

The Java™ Servlet API includes a `Cookie` class that you can use to set or retrieve the cookie from the HTTP header. HTTP cookies include a name and value pair.

The `startSession` method shown here is in the [LoginServlet](#) program. In this method, the name in the name and value pair used to create the `Cookie` is `JDCAUCTION`, and a unique identifier generated by the server is the value.

```
protected Session startSession(String theuser,
    String password,
        HttpServletResponse response) {
    Session session = null;
    if ( verifyPassword(theuser, password) ) {
    // Create a session
        session = new Session (theuser);
        session.setExpires (sessionTimeout + i
            System.currentTimeMillis());
        sessionCache.put (session);

    // Create a client cookie
        Cookie c = new Cookie("JDCAUCTION",
            String.valueOf(session.getId()));
        c.setPath ("/");
        c.setMaxAge (-1);
        c.setDomain (domain);
        response.addCookie (c);
    }
    return session;
}
```

Later versions of the Servlet API include a `Session` API, to create a session using the Servlet API in the previous example you can use the `getSession` method.

```
HttpSession session = new Session (true);
```

The `startSession` method is called by requesting the login action from a POST to the `LoginServlet` as follows:

```
<FORM ACTION="/LoginServlet" METHOD="POST">
```

```

<TABLE>
<INPUT TYPE="HIDDEN" NAME="action" VALUE="login">
<TR>
<TD>Enter your user id:</TD>
<TD><INPUT TYPE="TEXT" SIZE=20
    NAME="theuser"></TD>
</TR>
<TR>
<TD>Enter your password:<TD>
<TD><INPUT TYPE="PASSWORD" SIZE=20
    NAME="password"></TD>
</TR>
</TABLE>
<INPUT TYPE="SUBMIT" VALUE="Login" NAME="Enter">
</FORM>

```

The cookie is created with an maximum age of -1, which means the cookie is not stored but remains alive while the browser runs. The value is set in seconds, although when using values smaller than a few minutes you need to be careful of machine times being slightly out of sync.

The path value can be used to specify that the cookie only applies to files and directories under the path set on that machine. In this example the root path / means the cookie is applicable to all directories.

The domain value in the example is read from the initialization parameters for the servlet. If the domain is null, the cookie is applied to that machines domain only.

Retrieving a Cookie

The cookie is retrieved from the HTTP headers with a call to the `getCookies` method on the request:

```
Cookie c[] = request.getCookies();
```

You can later retrieve the name and value pair settings by calling the `Cookie.getName` method to retrieve the name, and the `Cookie.getValue` method to retrieve the value.

`LoginServlet` has a `validateSession` method that checks the user's cookies to find a `JDCAUCTION` cookie that was set in this domain:

```

private Session validateSession
    (HttpServletRequest request,
     HttpServletResponse response) {
    Cookie c[] = request.getCookies();
    Session session = null;
    if( c != null ) {
        Hashtable sessionTable = new Hashtable();
        for (int i=0; i < c.length &&
            session == null; i++ ) {
            if(c[i].getName().equals("JDCAUCTION")) {

```

```

        String key = String.valueOf (c[i].getValue());
        session=sessionCache.get(key);
    }
}
return session;
}

```

If you use the Servlet session API then you can use the following method, note that the parameter is false to specify the session value is returned and that a new session is not created.

```
HttpSession session = request.getSession(false);
```

Generating Sessions

The `LoginServlet.validateSession` method returns a `Session` object represented by the [Session](#) class. The `Session` class uses an identifier generated from a numeric sequence. This numbered session identifier is the value part of the name and value pair stored in the cookie.

The only way to reference the user name on the server is with this session identifier, which is stored in a simple memory cache with the other session IDs. When a user terminates a session, the `LoginServlet` `logout` action is called like this:

```
http://localhost:7001/LoginServlet?action=logout
```

The session cache implemented in the [SessionCache.java](#) program includes a reaper thread to remove sessions older than a preset time. The preset timeout could be measured in hours or days depending on how many visitors visit the site.

Preventing Page Caching

The `LoginServlet.setNoCache` method sets the `Cache-Control` or `Pragma` values (depending on which version of the HTTP protocol is being used) in the response header to `no-cache`. The expiration header `Expires` is also set to 0, alternatively you can set the time to be the current system time. Even if the client does not cache the page, there are often proxy servers in a corporate network that would. Only pages using Secure Socket Layer (SSL) are not cached by default.

```

private void setNoCache (HttpServletRequest request,
                        HttpServletResponse response) {
    if(request.getProtocol().compareTo ("HTTP/1.0") == 0) {
        response.setHeader ("Pragma", "no-cache");
    } else if (request.getProtocol().compareTo
               ("HTTP/1.1") == 0) {
        response.setHeader ("Cache-Control", "no-cache");
    }
}

```

```

    }
    response.setDateHeader ("Expires", 0);
}

```

Restricting Access and Redirections

If you install the `LoginServlet` as the default servlet or servlet to run when serving any page under the document root, you can use cookies to restrict users to certain sections of the site. For example, you can allow users who have cookies that state they have logged in to access sections of the site that require a login password and keep all others out.

The [LoginServlet](#) program checks for a restricted directory in its `init` method. The `init` method shown below sets the `protectedDir` variable to `true` if the `config` variable passed to it specifies a protected directory. The web server configuration file provides the settings passed to a servlet in the `config` variable.

```

public void init(ServletConfig config)
                throws ServletException {
    super.init(config);
    domain = config.getInitParameter("domain");
    restricted = config.getInitParameter("restricted");
    if(restricted != null) {
        protectedDir=true;
    }
}

```

Later on in the `validateSession` and `service` methods, the `protectedDir` variable is checked and the `HttpServletResponse.sendRedirect` method is called to send the user to the correct page based on their login and session status.

```

    if(protectedDir) {
        response.sendRedirect (restricted+"/index.html");
    }else{
        response.sendRedirect (defaultPage);
    }
}

```

The `init` method also retrieves the servlet context for the `FileServlet` servlet so methods can be called on the `FileServlet` in the `validateSession` method. The advantage to calling methods on the `FileServlet` servlet to serve the files rather than serving the files from within the `LoginServlet` servlet, is you get the full advantage of all the functionality added into the `FileServlet` servlet such as memory mapping or file caching. The downside is that the code may not be portable to other servers that do not have a `FileServlet` servlet. This code retrieves the `FileServlet` context.

```

FileServlet fileServlet=(FileServlet)
    config.getServletContext().getServlet("file");

```

The `validateSession` method prevents users without a logon session from accessing the restricted directory.

HTTP Error Codes

You can return a HTTP error code using the `sendError` method. For example, the HTTP 500 error code indicates an internal server error, and the 404 error code indicates page not found. This code segment returns the HTTP 500 error code.

```
protected void service (HttpServletRequest request,
                        HttpServletResponse response)
    throws ServletException {
    response.sendError (500);
}
```

Reading GET and POST Values

The Servlet API has a `getParameter` method in the `HttpServletRequest` class that returns the GET or POST value for the name you supply.

The HTTP GET request handles name and value pairs as part of the URL. The `getParameter` method parses the URL passed in, retrieves the `name=value` pairs delimited by the ampersand (&) character, and returns the value.

The HTTP POST request reads the name and value pairs from the input stream from the client. The `getParameter` method parses the input stream for the name and value pairs.

The `getParameter` method works well for simple servlets, but if you need to retrieve the POST parameters in the order they were placed on the web page or handle multi-part posts, you can write your own code to parse the input stream.

The next example returns POST parameters in the order they were received from the web page. Normally, the parameters are stored in a `Hashtable` which does not maintain the sequence order of elements stored in it. The example keeps a reference to each name and value pair in a vector that can be traversed to return the values in the order they were received by the server.

```
package auction;

import java.io.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;
```

```

public class PostServlet extends HttpServlet {
    private Vector paramOrder;
    private Hashtable parameters;

    public void init(ServletConfig config)
        throws ServletException {
        super.init(config);
    }

    public void service(HttpServletRequest request,
        HttpServletResponse response)
        throws IOException {

        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        if(request.getMethod().equals("POST")
            && request.getContentType().equals(
                "application/x-www-form-urlencoded")) {

            parameters=parsePostData(
                request.getContentLength(),
                request.getInputStream());
        }

        for(int i=0;i<paramOrder.size();i++) {
            String name=(String)paramOrder.elementAt(i);
            String value=getParameter((
                String)paramOrder.elementAt(i));
            out.println("name="+name+" value="+value);
        }
        out.println("</body></html>");
        out.close();
    }

    private Hashtable parsePostData(int length,
        ServletInputStream instream) {
        String valArray[] = null;
        int inputLen, offset;
        byte[] postedBytes = null;
        boolean dataRemaining=true;
        String postedBody;
        Hashtable ht = new Hashtable();
        paramOrder= new Vector(10);
        StringBuffer sb = new StringBuffer();

        if (length <=0) {
            return null;
        }
        postedBytes = new byte[length];
        try {
            offset = 0;
            while(dataRemaining) {
                inputLen = instream.read (postedBytes,
                    offset,
                    length - offset);
                if (inputLen <= 0) {
                    throw new IOException ("read error");
                }
                offset += inputLen;
                if((length-offset) ==0) {

```

```

        dataRemaining=false;
    }
}
} catch (IOException e) {
    System.out.println("Exception =" +e);
    return null;
}

postedBody = new String (postedBytes);
StringTokenizer st =
    new StringTokenizer(postedBody, "&");

String key=null;
String val=null;

while (st.hasMoreTokens()) {
    String pair = (String)st.nextToken();
    int pos = pair.indexOf('=');
    if (pos == -1) {
        throw new IllegalArgumentException();
    }
    try {
        key = java.net.URLDecoder.decode(
            pair.substring(0, pos));
        val = java.net.URLDecoder.decode(
            pair.substring(pos+1,
                pair.length()));
    } catch (Exception e) {
        throw new IllegalArgumentException();
    }
    if (ht.containsKey(key)) {
        String oldVals[] = (String []) ht.get(key);
        valArray = new String[oldVals.length + 1];
        for (int i = 0; i < oldVals.length; i++) {
            valArray[i] = oldVals[i];
        }
        valArray[oldVals.length] = val;
    } else {
        valArray = new String[1];
        valArray[0] = val;
    }
    ht.put(key, valArray);
    paramOrder.addElement(key);
}
return ht;
}

public String getParameter(String name) {
    String vals[] = (String []) parameters.get(name);
    if (vals == null) {
        return null;
    }
    String vallist = vals[0];
    for (int i = 1; i < vals.length; i++) {
        vallist = vallist + "," + vals[i];
    }
    return vallist;
}
}
}

```

To find out whether the request is POST or GET, call the `getMethod`

in the `HttpServletRequest` class. To determine the format of the data being posted, call the `getContentType` method in the `HttpServletRequest` class. For simple HTML web pages, the type returned by this call will be `application/x-www-form-urlencoded`.

If you need to create a post with more than one part such as the one created by the following HTML form, the servlet will need to read the input stream from the post to reach individual section. Each section distinguished by a boundary defined in the post header.

```
<FORM ACTION="/PostMultiServlet"
  METHOD="POST" ENCTYPE="multipart/form-data">
<INPUT TYPE="TEXT" NAME="desc" value="">
<INPUT TYPE="FILE" NAME="filecontents" value="">
<INPUT TYPE="SUBMIT" VALUE="Submit" NAME="Submit">
</FORM>
```

The next example extracts a description and a file from the client browsers. It reads the input stream looking for a line matching the boundary string, reads the content line, skips a line and then reads the data associated with that part. The uploaded file is simply displayed, but could also be written to disk.

```
package auction;

import java.io.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class PostMultiServlet extends HttpServlet {

    public void init(ServletConfig config)
        throws ServletException {
        super.init(config);
    }

    public void service(HttpServletRequest request,
        HttpServletResponse response)
        throws ServletException, IOException {

        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        if (request.getMethod().equals("POST")
            && request.getContentType().startsWith(
                "multipart/form-data")) {

            int index = request.getContentType().indexOf(
                "boundary=");

            if (index < 0) {
                System.out.println("can't find boundary type");
                return;
            }

            String boundary =
                request.getContentType().substring(
                    index+9);
```

```

ServletInputStream instream =
    request.getInputStream();
byte[] tmpbuffer = new byte[8192];
int length=0;
String inputLine=null;
boolean moreData=true;

//Skip until form data is reached
length = instream.readLine(
    tmpbuffer,
    0,
    tmpbuffer.length);
inputLine = new String (tmpbuffer, 0, 0,
    length);

while(inputLine.indexOf(boundary)
    >0 && moreData) {
    length = instream.readLine(
        tmpbuffer,
        0,
        tmpbuffer.length);
    inputLine = new String (tmpbuffer, 0, 0,
        length);
    if(inputLine !=null)
        System.out.println("input="+inputLine);
    if(length<0) {
        moreData=false;
    }
}

if(moreData) {
    length = instream.readLine(
        tmpbuffer,
        0,
        tmpbuffer.length);
    inputLine = new String (tmpbuffer, 0, 0,
        length);

    if(inputLine.indexOf("desc") >=0) {
        length = instream.readLine(
            tmpbuffer,
            0,
            tmpbuffer.length);
        inputLine = new String (tmpbuffer, 0, 0,
            length);
        length = instream.readLine(
            tmpbuffer,
            0,
            tmpbuffer.length);
        inputLine = new String (tmpbuffer, 0, 0,
            length);
        System.out.println("desc="+inputLine);
    }
}

while(inputLine.indexOf(boundary)
    >0 && moreData) {
    length = instream.readLine(
        tmpbuffer,
        0,

```

```

        tmpbuffer.length);
        inputLine = new String (tmpbuffer, 0, 0,
                                length);
    }
    if(moreData) {
        length = instream.readLine(
            tmpbuffer,
            0,
            tmpbuffer.length);
        inputLine = new String (tmpbuffer, 0, 0,
                                length);

        if(inputLine.indexOf("filename") >=0) {
            int startindex=inputLine.indexOf(
                "filename");
            System.out.println("file name="+
                inputLine.substring(
                    startindex+10,
                    inputLine.indexOf("\"",
                    startindex+10)));
            length = instream.readLine(
                tmpbuffer,
                0,
                tmpbuffer.length);
            inputLine = new String (tmpbuffer, 0, 0,
                                    length);
        }
    }
    byte fileBytes[]=new byte[50000];
    int offset=0;
    if (moreData) {
        while(inputLine.indexOf(boundary)
            >0 && moreData) {
            length = instream.readLine(
                tmpbuffer,
                0,
                tmpbuffer.length);
            inputLine = new String (tmpbuffer, 0, 0, length);
            if(length>0 && (
                inputLine.indexOf(boundary) <0)) {
                System.arraycopy(
                    tmpbuffer,
                    0,
                    fileBytes,
                    offset,
                    length);
                offset+=length;
            } else {
                moreData=false;
            }
        }
    }
    // trim last two newline/return characters
    // before using data
    for(int i=0;i<offset-2;i++) {
        System.out.print((char)fileBytes[i]);
    }
}
out.println("</body></html>");
out.close();

```

```
}  
}
```

Threading

A servlet must be able to handle multiple concurrent requests. Any number of end users at any given time could invoke the servlet, and while the `init` method is always run single-threaded, the `service` method is multi-threaded to handle multiple requests.

This means any static or public fields accessed by the `service` method should be restricted to simple thread access. The example below uses the `synchronized` keyword to restrict access to a counter so it can only be updated by one thread at a time:

```
int counter  
Boolean lock = new Boolean(true);  
  
synchronized(lock){  
    counter++;  
}
```

HTTPS

Many servers, browsers, and the Java Plug-In have the ability to support the secure HTTP protocol called HTTPS. HTTPS is similar to HTTP except the data is transmitted over a secure socket layer (SSL) instead of a normal socket connection. Web servers often listen for HTTP requests on one port while listening for HTTPS requests on another.

The encrypted data that is sent over the network includes checks to verify if the data has been tampered in transit. SSL also authenticates the webserver to its clients by providing a public key certificate. In SSL 3.0 the client can also authenticate itself with the server, again using a public key certificate.

Public key cryptography (also called asymmetric key encryption) uses a public and private key pair. Any message encrypted (made unintelligible) with the private key in the pair can only be decrypted with the corresponding public key. Certificates are digitally signed statements generated from a trusted third party Certificate Authority. The Certificate Authority needs proof that you are who you say you are because clients will be trusting the certificate they receive. It is this certificate that contains the public key in the public and private key pair. The certificate is signed by the private key of the Certificate Authority, and most browsers know the public key for the main Certificate Authorities.

While public key encryption is good for authentication purposes, it is not as fast as symmetric key encryption and so the SSL protocol uses both types of keys in the lifecycle of an SSL connection. The client and server begin an HTTPS transaction with a connection initialization or handshaking phase.

It is in the handshaking stage that the server is authenticated using the certificate that the client has received. The client uses the server's public key to encrypt messages sent to the server. After the client has been authenticated and the encryption algorithm or cipher has been agreed between the two parties, new symmetric session keys are used to encrypt and decrypt any further communication.

The encryption algorithm or cipher can be one of many popular algorithms like Rivest Shamir and Adleman (RSA) or Data Encryption Standard (DES). The greater the number of bits used to make the key, the more difficult it is to break into using brute force search techniques.

HTTPS using public key cryptography and certificates lets you provide the amount of privacy your application needs for safe and secure transactions. Servers, browsers, and Java Plug-In have their own setup for enabling HTTPS using SSL communications. In general, the steps involve the following:

Get a private key and a digitally-signed certificate with the matching public key.

Install the certificate in a location specified by the software you are using (server, browser, or Java Plug-In).

Enable SSL features and specify your certificate and private key files as instructed in your documentation.

You should enable SSL features according to your specific application requirements depending on the level of security you need. For example, you do not need to verify the identity of customers browsing auction items, but you will want to encrypt credit card and other personal information supplied when buyers and sellers register to participate.

HTTPS can be used for any data not just HTTP web pages. Programs written in the Java language can be downloaded over an HTTPS connection, and you can open a connection to a HTTPS server in the Java Plug-in. To write a program in the Java language that uses SSL. SSL requires an SSL library and a detailed knowledge of the HTTPS handshaking process. Your SSL library

should cover the necessary steps as this information is restricted by export security control.

[\[TOP\]](#)

Printable Page 

[This page was updated: 4-Jun-2001]

[Products & APIs](#) | [Developer Connection](#) | [Docs & Training](#) | [Online Support](#)
[Community Discussion](#) | [Industry News](#) | [Solutions Marketplace](#) | [Case Studies](#)

[Glossary](#) | [Feedback](#) | [A-Z Index](#)

For more information on Java technology and other software from Sun Microsystems, call: (800) 786-7638
Outside the U.S. and Canada, dial your country's [AT&T Direct Access Number](#) first.



Copyright © 1995-2001 [Sun Microsystems, Inc.](#)
All Rights Reserved. [Terms of Use](#). [Privacy Policy](#).



Products & APIs
 Developer Connection
 Docs & Training
 Online Support
 Community Discussion
 Industry News
 Solutions Marketplace
 Case Studies

[Training Index](#)

Writing Advanced Applications

Chapter 5: JNI Technology

[<<BACK] [CONTENTS] [NEXT>>]

Printable Page 

Requires login

Early Access 
 Downloads

Bug Database 
 Submit a Bug
 View Database

Newsletters
 Back Issues
 Subscribe

Learning Centers
 Articles
 Bookshelf
 Code Samples
 New to Java
 Question of the Week
 Quizzes
 Tech Tips
 Tutorials

Forums

The Java™ platform is relatively new, which means there could be times when you will need to integrate programs written with the Java language with existing non-Java language services, API toolkits, and programs. The Java platform provides the Java Native Interface (JNI) to help ease this type of integration.

The JNI defines a standard naming and calling convention so the Java¹ virtual machine can locate and invoke native methods. In fact, JNI is built into the Java virtual machine so the Java virtual machine can invoke local system calls to perform input and output, graphics, networking, and threading operations on the host operating system.

This chapter explains how to use JNI in programs written in the Java language to call any libraries on the local machine, call Java language methods from inside native code, and how to create and run a Java VM instance. To show how you can put JNI to use, the examples in this chapter include integrating JNI with the Xbase C++ database API, and how you can call a mathematic function. [Xbase](#) has sources you can download.

[JNI Example](#)

[Strings and Arrays](#)

[Other Programming Issues](#)

In a Rush?

This table links you directly to specific topics.

Topic	Section

Technology Centers

JNI Example	About the Example Generate the Header File Method Signature Implement the Native Method Compile the Dynamic or Shared Object Library Run the Example
Strings, Arrays, and Fields	Passing Strings Passing Arrays Pinning Array Object Arrays Multi-Dimensional Arrays Accessing Fields
Other Programming Issues	Language Issues Calling Methods Accessing Fields Threads and Synchronization Memory Issues Invocation Attaching Threads

¹ As used on this web site, the terms "Java virtual machine" or "JVM" mean a virtual machine for the Java platform.

[\[TOP\]](#)

Printable Page 

[This page was updated: 4-Jun-2001]

[Products & APIs](#) | [Developer Connection](#) | [Docs & Training](#) | [Online Support](#)
[Community Discussion](#) | [Industry News](#) | [Solutions Marketplace](#) | [Case Studies](#)

[Glossary](#) | [Feedback](#) | [A-Z Index](#)

For more information on Java technology and other software from Sun Microsystems, call: (800) 786-7638
 Outside the U.S. and Canada, dial your country's [AT&T Direct Access Number](#) first.



Copyright © 1995-2001 [Sun Microsystems, Inc.](#)
 All Rights Reserved. [Terms of Use](#). [Privacy Policy](#).



Products & APIs
 Developer Connection
 Docs & Training
 Online Support
 Community Discussion
 Industry News
 Solutions Marketplace
 Case Studies

[Training Index](#)

Writing Advanced Applications

Chapter 5 Continued: JNI Example

[<<BACK] [CONTENTS] [NEXT>>]

Printable Page 

This section presents the `ReadFile` example program. This example shows how you can use the Java™ Native Interface (JNI) to invoke a native method that makes C function calls to map a file into memory.

■ Requires login

Early Access ■
 Downloads

Bug Database ■
 Submit a Bug
 View Database

Newsletters
 Back Issues
 Subscribe

Learning Centers
 Articles
 Bookshelf
 Code Samples
 New to Java
 Question of the Week
 Quizzes
 Tech Tips
 Tutorials

Forums

[About the Example](#)

- [Native Method Declaration](#)
- [Load the Library](#)
- [Compile the Program](#)

[Generate the Header File](#)

[Method Signature](#)

[Implement the Native Method](#)

[Compile the Dynamic or Shared Object Library](#)

[Run the Example](#)

About the Example

You can call code written in any programming language from a program written in the Java language by declaring a native Java method, loading the library that contains the native code, and then calling the native method. The `ReadFile` source code below does exactly this.

However, successfully running the program requires a few additional steps beyond compiling the Java language source file. After you compile, but before you run the example, you have to generate a header file. The native code implements the function definitions contained in the generated header file and implements the business logic as well. The following sections walk through all the steps.

```
import java.util.*;
```

Technology Centers

```
class ReadFile {
//Native method declaration
    native byte[] loadFile(String name);
//Load the library
    static {
        System.loadLibrary("nativelib");
    }

    public static void main(String args[]) {
        byte buf[];
//Create class instance
        ReadFile mappedFile=new ReadFile();
//Call native method to load ReadFile.java
        buf=mappedFile.loadFile("ReadFile.java");
//Print contents of ReadFile.java
        for(int i=0;i<buf.length;i++) {
            System.out.print((char)buf[i]);
        }
    }
}
```

Native Method Declaration

The `native` declaration provides the bridge to run the native function in the Java¹ virtual machine. In this example, the `loadFile` function maps onto a C function called `Java_ReadFile_loadFile`. The function implementation accepts a `String` that represents a file name and returns the contents of that file in the byte array.

```
    native byte[] loadFile(String name);
```

Load the Library

The library containing the native code implementation is loaded by a call to `System.loadLibrary()`. Placing this call in a static initializer ensures this library is only loaded once per class. The library can be loaded outside of the static block if your application requires it. You might need to configure your environment so the `loadLibrary` method can find your native code library.

```
    static {
        System.loadLibrary("nativelib");
    }
```

Compile the Program

To compile the program, just run the `javac` compiler command as you normally would:

```
javac ReadFile.java
```

Next, you need to generate a header file with the native method declaration and implement the native method to call the C functions for loading and reading a file.

Generate the Header File

To generate a header file, run the `javah` command on the `ReadFile` class. In this example, the generated header file is named `ReadFile.h`. It provides a method signature that you have to use when you implement the `loadfile` native function.

```
javah -jni ReadFile
```

Note: When running `javah` on your own classes, be sure to use the fully-qualified class name.

Method Signature

The `ReadFile.h` header file defines the interface to map the Java language method to the native C function. It uses a method signature to map the arguments and return value of the Java language `mappedfile.loadFile` method to the `loadFile` native method in the `nativelib` library. Here is the `loadFile` native method mapping (method signature):

```
/*
 * Class:      ReadFile
 * Method:     loadFile
 * Signature:  (Ljava/lang/String;)[B
 */
JNIEXPORT jbyteArray JNICALL Java_ReadFile_loadFile
    (JNIEnv *, jobject, jstring);
```

The method signature parameters function as follows:

`JNIEnv *`: A pointer to the JNI environment. This pointer is a handle to the current thread in the Java virtual machine, and contains mapping and other housekeeping information.

`jobject`: A reference to the method that called this native code. If the calling method is static, this parameter would be type `jclass` instead of `jobject`.

`jstring`: The parameter supplied to the native method. In this example, it is the name of the file to be read.

Implement the Native Method

In this native C source file, the `loadFile` definition is a copy and paste of the C declaration contained in `ReadFile.h`. The definition is followed by the native method implementation. JNI provides a mapping for both C and C++ by default.

```

JNIEXPORT jbyteArray JNICALL Java_ReadFile_loadFile
(JNIEnv * env, jobject jobj, jstring name) {
    caddr_t m;
    jbyteArray jb;
    jboolean iscopy;
    struct stat finfo;
    const char *mfile = (*env)->GetStringUTFChars(
        env, name, &iscopy);
    int fd = open(mfile, O_RDONLY);

    if (fd == -1) {
        printf("Could not open %s\n", mfile);
    }
    lstat(mfile, &finfo);
    m = mmap((caddr_t) 0, finfo.st_size,
        PROT_READ, MAP_PRIVATE, fd, 0);
    if (m == (caddr_t)-1) {
        printf("Could not mmap %s\n", mfile);
        return(0);
    }
    jb = (*env)->NewByteArray(env, finfo.st_size);
    (*env)->SetByteArrayRegion(env, jb, 0,
        finfo.st_size, (jbyte *)m);
    close(fd);
    (*env)->ReleaseStringUTFChars(env, name, mfile);
    return (jb);
}

```

You can approach calling an existing C function instead of implementing one, in one of two ways:

1. Map the name generated by JNI to the existing C function name. The [Language Issues](#) section shows how to map between Xbase database functions and Java language code
2. Use the shared stubs code available from the [JNI page](#) on the java.sun.com web site.

Compile the Dynamic or Shared Object Library

The library needs to be compiled as a dynamic or shared object library so it can be loaded at runtime. Static or archive libraries are compiled into an executable and cannot be loaded at runtime. The shared object or dynamic library for the `loadFile` example is compiled as follows:

Gnu C/Linux:

```

gcc -o libnativelib.so -shared -Wl,-soname,libnative.so
-I/export/home/jdk1.2/include
-I/export/home/jdk1.2/include/linux nativelib.c
-static -lc

```

Gnu C++/Linux with Xbase

```
g++ -o libdbmaplib.so -shared -Wl,-soname,libdbmap.so  
-I/export/home/jdk1.2/include  
-I/export/home/jdk1.2/include/linux  
dbmaplib.cc -static -lc -lxbase
```

Win32/WinNT/Win2000

```
cl -Ic:/jdk1.2/include  
-Ic:/jdk1.2/include/win32  
-LD nativelib.c -Felibnative.dll
```

Run the Example

To run the example, the Java virtual machine needs to be able to find the native library. To do this, set the library path to the current directory as follows:

Unix or Linux:

```
LD_LIBRARY_PATH=`pwd`  
export LD_LIBRARY_PATH
```

Windows NT/2000/95:

```
set PATH=%path%;.
```

With the library path properly specified for your platform, invoke the program as you normally would with the interpreter command:

```
java ReadFile
```

¹ As used on this web site, the terms "Java virtual machine" or "JVM" mean a virtual machine for the Java platform.

[\[TOP\]](#)

Printable Page 

[This page was updated: 4-Jun-2001]

[Products & APIs](#) | [Developer Connection](#) | [Docs & Training](#) | [Online Support](#)
[Community Discussion](#) | [Industry News](#) | [Solutions Marketplace](#) | [Case Studies](#)

[Glossary](#) | [Feedback](#) | [A-Z Index](#)

For more information on Java technology
and other software from Sun Microsystems, call:
(800) 786-7638
Outside the U.S. and Canada, dial your country's
[AT&T Direct Access Number](#) first.



Copyright © 1995-2001 [Sun Microsystems, Inc.](#)
All Rights Reserved. [Terms of Use](#). [Privacy Policy](#).



[Products & APIs](#)
[Developer Connection](#)
[Docs & Training](#)
[Online Support](#)
[Community Discussion](#)
[Industry News](#)
[Solutions Marketplace](#)
[Case Studies](#)

[Training Index](#)

Writing Advanced Applications

Chapter 5 Continued: Strings and Arrays

[\[<<BACK\]](#)
[\[CONTENTS\]](#)
[\[NEXT>>\]](#)

This section explains how to pass string and array data between a program written in the Java™ programming language and other languages.

Printable Page 

■ Requires login

Early Access ■
[Downloads](#)

Bug Database ■
[Submit a Bug](#)
[View Database](#)

Newsletters
[Back Issues](#)
[Subscribe](#)

Learning Centers
[Articles](#)
[Bookshelf](#)
[Code Samples](#)
[New to Java](#)
[Question of the Week](#)
[Quizzes](#)
[Tech Tips](#)
[Tutorials](#)

[Forums](#)

[Passing Strings](#)

[Passing Arrays](#)

[Pinning Array](#)

[Object Arrays](#)

[Multi-Dimensional Arrays](#)

Passing Strings

The `String` object in the Java language, which is represented as `jstring` in Java Native Interface (JNI), is a 16 bit unicode string. In C a string is by default constructed from 8 bit characters. So, to access a Java language `String` object passed to a C or C++ function or return a C or C++ string to a Java language method, you need to use JNI conversion functions in your native method implementation.

The `GetStringUTFChar` function retrieves 8-bit characters from a 16-bit `jstring` using the Unicode Transformation Format (UTF). UTF represents Unicode as a string of 8 or 16 bit characters without losing any information. The third parameter `GetStringUTFChar` results the result `JNI_TRUE` if it made a local copy of the `jstring` or `JNI_FALSE` otherwise.

C Version:

```
(*env)->GetStringUTFChars(env, name, iscopy)
```

C++ Version:

```
env->GetStringUTFChars(name, iscopy)
```

The following C JNI function converts an array of C characters to a

Technology Centers `jstring`:

```
(*env)->NewStringUTF(env, lastfile)
```

The example below converts the `lastfile[80]` C character array to a `jstring`, which is returned to the calling Java language method:

```
static char lastfile[80];

JNIEXPORT jstring JNICALL Java_ReadFile_lastFile
    (JNIEnv *env, jobject obj) {
    return((*env)->NewStringUTF(env, lastfile));
}
```

To let the Java¹ virtual machine know you are finished with the UTF representation, call the `ReleaseStringUTFChars` conversion function as shown below. The second argument is the original `jstring` value used to construct the UTF representation, and the third argument is the reference to the local representation of that String.

```
(*env)->ReleaseStringUTFChars(env, name, mfile);
```

If your native code can work with Unicode, without needing the intermediate UTF representation, call the `GetStringChars` function to retrieve the unicode string, and release the reference with a call to `ReleaseStringChars`:

```
JNIEXPORT jbyteArray JNICALL Java_ReadFile_loadFile
    (JNIEnv * env, jobject obj, jstring name) {
    caddr_t m;
    jbyteArray jb;
    struct stat finfo;
    jboolean iscopy;
    const jchar *mfile = (*env)->GetStringChars(env,
        name, &iscopy);
    //...
    (*env)->ReleaseStringChars(env, name, mfile);
```

Passing Arrays

In the example presented in the last section, the `loadFile` native method returns the contents of a file in a byte array, which is a primitive type in the Java programming language. You can retrieve and create primitive types in the Java language by calling the appropriate `TypeArray` function.

For example, to create a new array of floats, call `NewFloatArray`, or to create a new array of bytes, call `NewByteArray`. This naming scheme extends to retrieving elements from, adding elements to, and changing elements in the array. To get a new array of bytes, call `GetByteArrayElements`. To add elements to or change elements in the array, call `Set<type>ArrayElements`.

The `GetByteArrayElements` function affects the entire array. To work on a portion of the array, call `GetByteArrayRegion` instead. There is only a `Set<type>ArrayRegion` function for changing array elements. However the region could be of size 1, which is equivalent to the non-existent `Set<type>ArrayElements`.

Native Code Type	Functions used
jboolean	<code>NewBooleanArray</code>
	<code>GetBooleanArrayElements</code>
	<code>GetBooleanArrayRegion/SetBooleanArrayRegion</code>
	<code>ReleaseBooleanArrayRegion</code>
jbyte	<code>NewByteArray</code>
	<code>GetByteArrayElements</code>
	<code>GetByteArrayRegion/SetByteArrayRegion</code>
	<code>ReleaseByteArrayRegion</code>
jchar	<code>NewCharArray</code>
	<code>GetCharArrayElements</code>
	<code>GetCharArrayRegion/SetCharArrayRegion</code>
	<code>ReleaseCharArrayRegion</code>
jdouble	<code>NewDoubleArray</code>
	<code>GetDoubleArrayElements</code>
	<code>GetDoubleArrayRegion/SetDoubleArrayRegion</code>
	<code>ReleaseDoubleArrayRegion</code>
jfloat	<code>NewFloatArray</code>
	<code>GetFloatArrayElements</code>
	<code>GetFloatArrayRegion/SetFloatArrayRegion</code>
	<code>ReleaseFloatArrayRegion</code>
jint	<code>NewIntArray</code>
	<code>GetIntArrayElements</code>
	<code>GetIntArrayRegion/SetIntArrayRegion</code>
	<code>ReleaseIntArrayRegion</code>
jlong	<code>NewLongArray</code>
	<code>GetLongArrayElements</code>
	<code>GetLongArrayRegion/SetLongArrayRegion</code>
	<code>ReleaseLongArrayRegion</code>
jobject	<code>NewObjectArray</code>

	GetObjectArrayElement/SetObjectArrayElement
jshort	NewShortArray
	GetShortArrayElements
	GetShortArrayRegion/SetShortArrayRegion
	ReleaseShortArrayRegion

In the `loadFile` native method from the example in the previous section, the entire array is updated by specifying a region that is the size of the file being read in:

```
jbyteArray jb;

jb=(*env)->NewByteArray(env, finfo.st_size);
(*env)->SetByteArrayRegion(env, jb, 0,
    finfo.st_size, (jbyte *)m);
close(fd);
```

The array is returned to the calling Java language method, which in turn, garbage collects the reference to the array when it is no longer used. The array can be explicitly freed with the following call.

```
(*env)-> ReleaseByteArrayElements(env, jb,
    (jbyte *)m, 0);
```

The last argument to the `ReleaseByteArrayElements` function above can have the following values:

0: Updates to the array from within the C code are reflected in the Java language copy.

JNI_COMMIT: The Java language copy is updated, but the local `jbyteArray` is not freed.

JNI_ABORT: Changes are not copied back, but the `jbyteArray` is freed. The value is used only if the array is obtained with a get mode of `JNI_TRUE` meaning the array is a copy.

Pinning Array

When retrieving an array, you can specify if this is a copy (`JNI_TRUE`) or a reference to the array residing in your Java language program (`JNI_FALSE`). If you use a reference to the array, you will want the array to stay where it is in the Java heap and not get moved by the garbage collector when it compacts heap memory. To prevent the array references from being moved, the Java virtual machine pins the array into memory. Pinning the array ensures that when the array is released, the correct elements are updated in the Java VM.

In the `loadfile` native method example from the previous section, the array is not explicitly released. One way to ensure the array is garbage collected when it is no longer needed is to call a Java language method, pass the byte array instead, and then free the local array copy. This technique is shown in the section on [Multi-Dimensional Arrays](#).

Object Arrays

You can store any Java language object in an array with the `NewObjectArray` and `SetObjectArrayElement` function calls. The main difference between an object array and an array of primitive types is that when constructing a `jobjectarray` type, the Java language class is used as a parameter.

This next C++ example shows how to call `NewObjectArray` to create an array of `String` objects. The size of the array is set to five, the class definition is returned from a call to `FindClass`, and the elements of the array are initialized with an empty string. The elements of the array are updated by calling `SetObjectArrayElement` with the position and value to put in the array.

```
#include <jni.h>
#include "ArrayHandler.h"

JNIEXPORT jobjectArray JNICALL
    Java_ArrayHandler_returnArray
(JNIEnv *env, jobject jobj){

    jobjectArray ret;
    int i;

    char *message[5]= {"first",
        "second",
        "third",
        "fourth",
        "fifth"};

    ret= (jobjectArray)env->NewObjectArray(5,
        env->FindClass("java/lang/String"),
        env->NewStringUTF(""));

    for(i=0;i<5;i++) {
        env->SetObjectArrayElement(
            ret,i,env->NewStringUTF(env, message[i]));
    }
    return(ret);
}
```

The Java class that calls this native method is as follows:

```
public class ArrayHandler {
    public native String[] returnArray();
    static{
        System.loadLibrary("nativelib");
    }
}
```

```

    }

    public static void main(String args[]) {
        String ar[];
        ArrayHandler ah= new ArrayHandler();
        ar = ah.returnArray();
        for (int i=0; i<5; i++) {
            System.out.println("array element"+i+
                               "=" + ar[i]);
        }
    }
}

```

Multi-Dimensional Arrays

You might need to call existing numerical and mathematical libraries such as the linear algebra library CLAPACK/LAPACK or other matrix crunching programs from your Java language program using native methods. Many of these libraries and programs use two-dimensional and higher order arrays.

In the Java programming language, any array that has more than one dimension is treated as an array of arrays. For example, a two-dimensional integer array is handled as an array of integer arrays. The array is read horizontally, or what is also termed as row order.

Other languages such as FORTRAN use column ordering so extra care is needed if your program hands a Java language array to a FORTRAN function. Also, the array elements in an application written in the Java programming language are not guaranteed to be contiguous in memory. Some numerical libraries use the knowledge that the array elements are stored next to each other in memory to perform speed optimizations, so you might need to make an additional local copy of the array to pass to those functions.

The next example passes a two-dimensional array to a native method which then extracts the elements, performs a calculation, and calls a Java language method to return the results.

The array is passed as an object array that contains an array of `jints`. The individual elements are extracted by first retrieving a `jintArray` instance from the object array by calling `GetObjectArrayElement`, and then extracting the elements from the `jintArray` row.

The example uses a fixed size matrix. If you do not know the size of the array being used, the `GetArrayLength(array)` function returns the size of the outermost array. You will need to call the

`GetArrayLength(array)` function on each dimension of the array to discover the total size of the array.

The new array sent back to the program written in the Java language is built in reverse. First, a `jintArray` instance is created and that instance is set in the object array by calling `SetObjectArrayElement`.

```
public class ArrayManipulation {
    private int arrayResults[][];
    Boolean lock=new Boolean(true);
    int arraySize=-1;

    public native void manipulateArray(
        int[][] multiplier, Boolean lock);

    static{
        System.loadLibrary("nativelib");
    }

    public void sendArrayResults(int results[][]){
        arraySize=results.length;
        arrayResults=new int[results.length][];
        System.arraycopy(results,0,arrayResults,
            0,arraySize);
    }

    public void displayArray() {
        for (int i=0; i<arraySize; i++) {
            for(int j=0; j <arrayResults[i].length;j++) {
                System.out.println("array element "+i+", "+j+
                    "= " + arrayResults[i][j]);
            }
        }
    }

    public static void main(String args[]) {
        int[][] ar = new int[3][3];
        int count=3;
        for(int i=0;i<3;i++) {
            for(int j=0;j<3;j++) {
                ar[i][j]=count;
            }
            count++;
        }
        ArrayManipulation am= new ArrayManipulation();
        am.manipulateArray(ar, am.lock);
        am.displayArray();
    }
}

#include <jni.h>
#include <iostream.h>
#include "ArrayManipulation.h"

JNIEXPORT void
    JNICALL Java_ArrayManipulation_manipulateArray
    (JNIEnv *env, jobject jobj, jobjectArray elements,
        jobject lock){
```

```

    jobjectArray ret;
    int i,j;
    jint arraysize;
    int asize;
    jclass cls;
    jmethodID mid;
    jfieldID fid;
    long localArrayCopy[3][3];
    long localMatrix[3]={4,4,4};

    for(i=0; i<3; i++) {
        jintArray oneDim=
            (jintArray)env->GetObjectArrayElement(
                elements, i);
        jint *element=env->GetIntArrayElements(oneDim, 0);
        for(j=0; j<3; j++) {
            localArrayCopy[i][j]= element[j];
        }
    }

    // With the C++ copy of the array,
    // process the array with LAPACK, BLAS, etc.

    for (i=0;i<3;i++) {
        for (j=0; j<3 ; j++) {
            localArrayCopy[i][j]=
                localArrayCopy[i][j]*localMatrix[i];
        }
    }

    // Create array to send back
    jintArray row= (jintArray)env->NewIntArray(3);
    ret=(jobjectArray)env->NewObjectArray(
        3, env->GetObjectClass(row), 0);

    for(i=0;i<3;i++) {
        row= (jintArray)env->NewIntArray(3);
        env->SetIntArrayRegion((jintArray)row,(
            jsize)0,3,(jint *)localArrayCopy[i]);
        env->SetObjectArrayElement(ret,i,row);
    }

    cls=env->GetObjectClass(jobj);
    mid=env->GetMethodID(cls, "sendArrayResults",
        "([[I)V");

    if (mid == 0) {
        cout <<"Can't find method sendArrayResults";
        return;
    }


    env->ExceptionClear();
    env->MonitorEnter(lock);
    env->CallVoidMethod(jobj, mid, ret);
    env->MonitorExit(lock);
    if(env->ExceptionOccurred()) {
        cout << "error occured copying array back" << endl;
        env->ExceptionDescribe();
        env->ExceptionClear();
    }
    fid=env->GetFieldID(cls, "arraySize", "I");

```

```
if (fid == 0) {
    cout <<"Can't find field arraySize";
    return;
}
asize=env->GetIntField(jobj,fid);
if(!env->ExceptionOccurred()) {
    cout<< "Java array size=" << asize << endl;
} else {
    env->ExceptionClear();
}
return;
}
```

¹ As used on this web site, the terms "Java virtual machine" or "JVM" mean a virtual machine for the Java platform.

[\[TOP\]](#)

Printable Page 

[This page was updated: 4-Jun-2001]

[Products & APIs](#) | [Developer Connection](#) | [Docs & Training](#) | [Online Support](#)
[Community Discussion](#) | [Industry News](#) | [Solutions Marketplace](#) | [Case Studies](#)

[Glossary](#) | [Feedback](#) | [A-Z Index](#)

For more information on Java technology
and other software from Sun Microsystems, call:
(800) 786-7638
Outside the U.S. and Canada, dial your country's
[AT&T Direct Access Number](#) first.



Copyright © 1995-2001 [Sun Microsystems, Inc.](#)
All Rights Reserved. [Terms of Use.](#) [Privacy Policy.](#)



Products & APIs
 Developer Connection
 Docs & Training
 Online Support
 Community Discussion
 Industry News
 Solutions Marketplace
 Case Studies

[Training Index](#)

Writing Advanced Applications

Chapter 5 Continued: Other Programming Issues

[<<BACK] [CONTENTS] [NEXT>>]

Printable Page 

■ Requires login

Early Access ■
 Downloads

Bug Database ■
 Submit a Bug
 View Database

Newsletters
 Back Issues
 Subscribe

Learning Centers
 Articles
 Bookshelf
 Code Samples
 New to Java
 Question of the Week
 Quizzes
 Tech Tips
 Tutorials

Forums

This section presents information on accessing classes, methods, and fields, and covers threading, memory, and Java¹ virtual machine issues.

[Language Issues](#)
[Calling Methods](#)
[Accessing Fields](#)
[Threads and Synchronization](#)
[Memory Issues](#)
[Invocation](#)
[Attaching Threads](#)

Language issues

So far, the native method examples have covered calling standalone C and C++ functions that either return a result or modify parameters passed into the function. However, C++ like the Java language uses instances of classes. If you create a class in one native method, the reference to this class does not have an equivalent class in the Java language, which makes it difficult to call functions on the C++ class that was first created.

One way to handle this situation is to keep a record of the C++ class reference and pass that back to a proxy or to the calling program. To ensure the C++ class persists across native method calls, use the C++ `new` operator to create a reference to the C++ object on the stack.

The following code provides a mapping between the Xbase database and Java language code. The Xbase database has a C++ API and uses an initialization class to perform subsequent database operations. When the class object is created, a pointer to this

Technology Centers object is returned as a Java language `int` value. You can use a long or larger value for machines with greater than 32 bits.

```
public class CallDB {
    public native int initdb();
    public native short opendb(String name, int ptr);
    public native short GetFieldNo(
        String fieldname, int ptr);

    static {
        System.loadLibrary("dbmaplib");
    }

    public static void main(String args[]) {
        String prefix=null;
        CallDB db=new CallDB();
        int res=db.initdb();
        if(args.length>=1) {
            prefix=args[0];
        }
        System.out.println(db.opendb("MYFILE.DBF", res));
        System.out.println(db.GetFieldNo("LASTNAME", res));
        System.out.println(db.GetFieldNo("FIRSTNAME", res));
    }
}
```

The return result from the call to the `initdb` native method, the `int` value, is passed to subsequent native method calls. The native code included in the `dbmaplib.cc` library de--references the Java language object passed in as a parameter and retrieves the object pointer. The line `xbDbf* Myfile=(xbDbf*)ptr;` casts the `int` pointer value to be a pointer of `Xbase` type `xbDbf`.

```
#include <jni.h>
#include <xbase/xbase.h>
#include "CallDB.h"

JNIEXPORT jint JNICALL Java_CallDB_initdb(
    JNIEnv *env, jobject obj) {
    xbXBase* x;
    x= new xbXBase();
    xbDbf* Myfile;
    Myfile =new xbDbf(x);
    return ((jint)Myfile);
}

JNIEXPORT jshort JNICALL Java_CallDB_opendb(
    JNIEnv *env, jobject obj,
    jstring dbname, jint ptr) {
    xbDbf* Myfile=(xbDbf*)ptr;
    return((*Myfile).OpenDatabase( "MYFILE.DBF"));
}

JNIEXPORT jshort JNICALL Java_CallDB_GetFieldNo(
    JNIEnv *env, jobject obj,
    jstring fieldname,
    jint ptr) {
    xbDbf* Myfile=(xbDbf*)ptr;
    return((*Myfile).GetFieldNo(
```



```

        env->GetStringUTFChars(fieldname,0));
    }

```

Calling Methods

The section on arrays highlighted some reasons for calling Java language methods from within native code; for example, when you need to free the result you intend to return. Other uses for calling Java native methods from within your native code would be if you need to return more than one result or you just simply want to modify Java language values from within native code.

Calling a Java language method from within native code involves the following three steps:

1. Retrieve a class reference
2. Retrieve a method identifier
3. Call the Methods

Retrieve a Class Reference

The first step is to retrieve a reference to the class that contains the methods you want to access. To retrieve a reference, you can either use the `FindClass` method or access the `jobject` or `jclass` argument to the native method.

Use the `FindClass` method:

```

JNIEXPORT void JNICALL Java_ArrayHandler_returnArray
(JNIEnv *env, jobject jobj){
    jclass cls = (*env)->FindClass(env, "ClassName");
}

```

Use the `jobject` argument:

```

JNIEXPORT void JNICALL Java_ArrayHandler_returnArray
(JNIEnv *env, jobject jobj){
    jclass cls=(*env)->GetObjectClass(env, jobj);
}

```

or

Use the `jclass` argument:

```

JNIEXPORT void JNICALL Java_ArrayHandler_returnArray
(JNIEnv *env, jclass jcls){
    jclass cls=jcls;
}

```

Retrieve a Method Identifier

Once the class has been obtained, the second step is to call the `GetMethodID` function to retrieve an identifier for a method you

select in the class. The identifier is needed when calling the method of that class instance. Because the Java language supports method overloading, you also need to specify the particular method signature you want to call. To find out what signature your Java language method uses, run the `javap` command as follows:

```
javap -s Class
```

The method signature used is displayed as a comment after each method declaration as shown here:

```
bash# javap -s ArrayHandler
Compiled from ArrayHandler.java
public class ArrayHandler extends java.lang.Object {
    java.lang.String arrayResults[];
    /* [Ljava/lang/String; */
    static {};
    /* ()V */
    public ArrayHandler();
    /* ()V */
    public void displayArray();
    /* ()V */
    public static void main(java.lang.String[]);
    /* ([Ljava/lang/String;)V */
    public native void returnArray();
    /* ()V */
    public void sendArrayResults(java.lang.String[]);
    /* ([Ljava/lang/String;)V */
}
```

Use the `GetMethodID` function to call instance methods in an object instance, or use the `GetStaticMethodID` function to call static method. Their argument lists are the same.

Call the Methods

Third, the matching instance method is called using a `Call<type>Method` function. The `type` value can be `Void`, `Object`, `Boolean`, `Byte`, `Char`, `Short`, `Int`, `Long`, `Float`, **OR** `Double`.

The parameters to the method can be passed as a comma-separated list, an array of values to the `Call<type>MethodA` function, or as a `va_list`. The `va_list` is a construct often used for variable argument lists in C. `CallMethodV` is the function used to pass a `va_list` ().

Static methods are called in a similar way except the method naming includes an additional `Static` identifier, `CallStaticByteMethodA`, and the `jclass` value is used instead of `jobject`.

The next example returns the object array by calling the `sendArrayResults` method from the `ArrayHandler` class.

```

// ArrayHandler.java
public class ArrayHandler {
    private String arrayResults[];
    int arraySize=-1;

    public native void returnArray();

    static{
        System.loadLibrary("nativelib");
    }

    public void sendArrayResults(String results[]) {
        arraySize=results.length;
        arrayResults=new String[arraySize];
        System.arraycopy(results,0,
                        arrayResults,0,arraySize);
    }

    public void displayArray() {
        for (int i=0; i<arraySize; i++) {
            System.out.println("array element
                "+i+ " = " + arrayResults[i]);
        }
    }

    public static void main(String args[]) {
        String ar[];
        ArrayHandler ah= new ArrayHandler();
        ah.returnArray();
        ah.displayArray();
    }
}

```

The native C++ code is defined as follows:

```

#include <jni.h>
#include <iostream.h>
#include "ArrayHandler.h"

JNIEXPORT void JNICALL Java_ArrayHandler_returnArray
(JNIEnv *env, jobject obj){

    jobjectArray ret;
    int i;
    jclass cls;
    jmethodID mid;

    char *message[5]= {"first",
        "second",
        "third",
        "fourth",
        "fifth"};

    ret=(jobjectArray)env->NewObjectArray(5,
        env->FindClass("java/lang/String"),
        env->NewStringUTF(""));

    for(i=0;i<5;i++) {
        env->SetObjectArrayElement(
            ret,i,env->NewStringUTF(message[i]));
    }
}

```

```

    cls=env->GetObjectClass(jobj);
    mid=env->GetMethodID(cls,
        "sendArrayResults",
        "([Ljava/lang/String;)V");
    if (mid == 0) {
        cout <<"Can't find method sendArrayResults";
        return;
    }

    env->ExceptionClear();
    env->CallVoidMethod(jobj, mid, ret);
    if(env->ExceptionOccurred()) {
        cout << "error occured copying array back" <<endl;
        env->ExceptionDescribe();
        env->ExceptionClear();
    }
    return;
}

```

To build this on Linux, run the following commands:

```

javac ArrayHandler.java
javah -jni ArrayHandler

g++ -o libnativelib.so
    -shared -Wl,-soname,libnative.so
    -I/export/home/jdk1.2/include
    -I/export/home/jdk1.2/include/linux nativelib.cc
    -lc

```

If you want to specify a super class method; for example, to call the parent constructor, you can do so by calling the `CallNonvirtual<type>Method` functions.

One important point when calling Java language methods or fields from within native code is you need to catch any raised exceptions. The `ExceptionClear` function clears any pending exceptions while the `ExceptionOccured` function checks to see if an exception has been raised in the current JNI session.

Accessing Fields

Accessing Java language fields from within native code is similar to calling Java language methods. However, the set or field is retrieved with a field ID, instead of a method ID.

The first thing you need to do is retrieve a field ID. You can use the `GetFieldID` function, but specify the field name and signature in place of the method name and signature. Once you have the field ID, call a `Get<type>Field` function to set the field value. The `<type>` is the same as the native type being returned except the `j` is dropped and the first letter is capitalized. For example, the `<type>` value is `Int` for native type `jint`, and `Byte` for native type `jbyte`.

The `Get<type>Field` function result is returned as the native type.

For example, to retrieve the `arraySize` field in the `ArrayHandler` class, call `GetIntField` as shown in the following example.

The field can be set by calling the `env->SetIntField(jobj, fid, arraysize)` functions. Static fields can be set by calling `SetStaticIntField(jclass, fid, arraysize)` and retrieved by calling `GetStaticIntField(jobj, fid)`.

```
#include <jni.h>
#include <iostream.h>
#include "ArrayHandler.h"

JNIEXPORT void JNICALL Java_ArrayHandler_returnArray
(JNIEnv *env, jobject jobj){

    jobjectArray ret;
    int i;
    jint arraysize;
    jclass cls;
    jmethodID mid;
    jfieldID fid;

    char *message[5]= {"first",
                      "second",
                      "third",
                      "fourth",
                      "fifth"};

    ret=(jobjectArray)env->NewObjectArray(5,
        env->FindClass("java/lang/String"),
        env->NewStringUTF(""));

    for(i=0;i<5;i++) {
        env->SetObjectArrayElement(
            ret,i,env->NewStringUTF(message[i]));
    }

    cls=env->GetObjectClass(jobj);
    mid=env->GetMethodID(cls,
        "sendArrayResults",
        "([Ljava/lang/String;)V");
    if (mid == 0) {
        cout <<"Can't find method sendArrayResults";
        return;
    }

    env->ExceptionClear();
    env->CallVoidMethod(jobj, mid, ret);
    if(env->ExceptionOccurred()) {
        cout << "error occured copying
                array back" << endl;
        env->ExceptionDescribe();
        env->ExceptionClear();
    }
    fid=env->GetFieldID(cls, "arraySize", "I");
    if (fid == 0) {
        cout <<"Can't find field arraySize";
        return;
    }
}
```

```

        arraysize=env->GetIntField(jobj, fid);
        if(!env->ExceptionOccurred()) {
            cout<< "size=" << arraysize << endl;
        } else {
            env->ExceptionClear();
        }
        return;
    }
}

```

Threads and Synchronization

Although the native library is loaded once per class, individual threads in an application written in the Java language use their own interface pointer when calling the native method. If you need to restrict access to a Java language object from within native code, you can either ensure that the Java language methods you call have explicit synchronization or you can use the JNI `MonitorEnter` and `MonitorExit` functions.

In the Java language, code is protected by a monitor whenever you specify the `synchronized` keyword. In the Java programming language, the monitor enter and exit routines are normally hidden from the application developer. In JNI, you need to explicitly delineate the entry and exit points of thread safe code.

The following example uses a `Boolean` object to restrict access to the `CallVoidMethod` function.

```

env->ExceptionClear();
env->MonitorEnter(lock);
env->CallVoidMethod(jobj, mid, ret);
env->MonitorExit(lock);
if(env->ExceptionOccurred()) {
    cout << "error occured copying array back" << endl;
    env->ExceptionDescribe();
    env->ExceptionClear();
}
}

```

You may find that in cases where you want access to a local system resource like a MFC window handle or message queue, it is better to use one Java `Thread` and access the local threaded native event queue or messaging system from within the native code.

Memory Issues

By default, JNI uses local references when creating objects inside a native method. This means when the method returns, the references are eligible to be garbage collected. If you want an object to persist across native method calls, use a global reference instead. A global reference is created from a local reference by calling `NewGlobalReference` on the the local reference.

You can explicitly mark a reference for garbage collection by calling `DeleteGlobalRef` on the reference. You can also create a weak style Global reference that is accessible outside the method, but can be garbage collected. To create one of these references, call `NewWeakGlobalRef` and `DeleteWeakGlobalRef` to mark the reference for garbage collection.

You can even explicitly mark a local reference for garbage collection by calling the `env->DeleteLocalRef(localobject)` method. This is useful if you are using a large amount of temporary data.

```
static jobject stringarray=0;

JNIEXPORT void JNICALL Java_ArrayHandler_returnArray
(JNIEnv *env, jobject jobj){

    jobjectArray ret;
    int i;
    jint arraysize;
    int asize;
    jclass cls, tmpcls;
    jmethodID mid;
    jfieldID fid;

    char *message[5]= {"first",
        "second",
        "third",
        "fourth",
        "fifth"};

    ret=(jobjectArray)env->NewObjectArray(5,
        env->FindClass("java/lang/String"),
        env->NewStringUTF(""));

    //Make the array available globally
    stringarray=env->NewGlobalRef(ret);

    //Process array
    // ...

    //clear local reference when finished..
    env->DeleteLocalRef(ret);
}
```

Invocation

The section on calling methods showed you how to call a method or field in a Java language program using the JNI interface and a class loaded using the `FindClass` function. With a little more code, you can create a standalone program that invokes a Java virtual machine and includes its own JNI interface pointer that can be used to create instances of Java language classes. In the Java 2 release, the runtime program named `java` is a small JNI application that does exactly that.

You can create a Java virtual machine with a call to `JNI_CreateJavaVM`, and shut the created Java virtual machine down with a call to `JNI_DestroyJavaVM`. A Java virtual machine might also need some additional environment properties. These properties can be passed to the `JNI_CreateJavaVM` function in a `JavaVMInitArgs` structure.

The `JavaVMInitArgs` structure contains a pointer to a `JavaVMOption` value used to store environment information such as the classpath and Java virtual machine version, or system properties that would normally be passed on the command line to the program.

When the `JNI_CreateJavaVM` function returns, you can call methods and create instances of classes using the `FindClass` and `NewObject` functions the same way you would for embedded native code.

Note: The Java virtual machine invocation used to be only used for native thread Java virtual machines. Some older Java virtual machines have a green threads option that is stable for invocation use. On a Unix platform, you may also need to explicitly link with `-lthread` or `-lpthread`.

This next program invokes a Java virtual machine, loads the `ArrayHandler` class, and retrieves the `arraySize` field which should contain the value minus one. The Java virtual machine options include the current path in the classpath and turning the Just-In-Time (JIT) compiler off `-Djava.compiler=NONE`.

```
#include <jni.h>

void main(int argc, char *argv[], char **envp) {
    JavaVMOption options[2];
    JavaVMInitArgs vm_args;
    JavaVM *jvm;
    JNIEnv *env;
    long result;
    jmethodID mid;
    jfieldID fid;
    jobject obj;
    jclass cls;
    int i, asize;

    options[0].optionString = ".";
    options[1].optionString = "-Djava.compiler=NONE";

    vm_args.version = JNI_VERSION_1_2;
    vm_args.options = options;
    vm_args.nOptions = 2;
    vm_args.ignoreUnrecognized = JNI_FALSE;
```



```

    result = JNI_CreateJavaVM(
        &jvm,(void **)&env, &vm_args);
    if(result == JNI_ERR ) {
        printf("Error invoking the JVM");
        exit (-1);
    }

    cls = (*env)->FindClass(env,"ArrayHandler");
    if( cls == NULL ) {
        printf("can't find class ArrayHandler\n");
        exit (-1);
    }
    (*env)->ExceptionClear(env);
    mid=(*env)->GetMethodID(env, cls, "<init>", "()V");
    jobj=(*env)->NewObject(env, cls, mid);
    fid=(*env)->GetFieldID(env, cls, "arraySize", "I");
    asize=(*env)->GetIntField(env, jobj, fid);

    printf("size of array is %d",asize);
    (*jvm)->DestroyJavaVM(jvm);
}

```

Attaching Threads

After the Java virtual machine is invoked, there is one local thread running the Java virtual machine. You can create more threads in the local operating system and attach the Java virtual machine to those new threads. You might want to do this if your native application is multi-threaded.

Attach the local thread to the Java virtual machine with a call to `AttachCurrentThread`. You need to supply pointers to the Java virtual machine instance and JNI environment. In the Java 2 platform, you can also specify in the third parameter the thread name and/or group you want this new thread to live under. It is important to detach any thread that has been previously attached; otherwise, the program will not exit when you call `DestroyJavaVM`.

```

#include <jni.h>
#include <pthread.h>

JavaVM *jvm;

void *native_thread(void *arg) {
    JNIEnv *env;
    jclass cls;
    jmethodID mid;
    jfieldID fid;
    jint result;
    jobject jobj;
    JavaVMAttachArgs args;
    jint asize;

    args.version= JNI_VERSION_1_2;
    args.name="user";
}

```

```

    args.group=NULL;
    result=(*jvm)->AttachCurrentThread(
        jvm, (void **)&env, &args);

    cls = (*env)->FindClass(env,"ArrayHandler");
    if( cls == NULL ) {
        printf("can't find class ArrayHandler\n");
        exit (-1);
    }
    (*env)->ExceptionClear(env);
    mid=(*env)->GetMethodID(env, cls, "<init>", "()"V");
    jobj=(*env)->NewObject(env, cls, mid);
    fid=(*env)->GetFieldID(env, cls, "arraySize", "I");
    asize=(*env)->GetIntField(env, jobj, fid);
    printf("size of array is %d\n",asize);
    (*jvm)->DetachCurrentThread(jvm);
}

void main(int argc, char *argv[], char **envp) {
    JavaVMOption *options;
    JavaVMInitArgs vm_args;
    JNIEnv *env;
    jint result;
    pthread_t tid;
    int thr_id;
    int i;

    options = (void *)malloc(3 * sizeof(JavaVMOption));

    options[0].optionString = "-Djava.class.path=";
    options[1].optionString = "-Djava.compiler=NONE";

    vm_args.version = JNI_VERSION_1_2;
    vm_args.options = options;
    vm_args.nOptions = 2;
    vm_args.ignoreUnrecognized = JNI_FALSE;

    result = JNI_CreateJavaVM(&jvm,(void **)&env, &vm_args);
    if(result == JNI_ERR ) {
        printf("Error invoking the JVM");
        exit (-1);
    }

    thr_id=pthread_create(&tid, NULL, native_thread, NULL);

    // If you don't have join, sleep instead
    //sleep(1000);
    pthread_join(tid, NULL);
    (*jvm)->DestroyJavaVM(jvm);
    exit(0);
}

```

¹ As used on this web site, the terms "Java virtual machine" or "JVM" mean a virtual machine for the Java platform.

[\[TOP\]](#)

Printable Page 

[This page was updated: 4-Jun-2001]

[Products & APIs](#) | [Developer Connection](#) | [Docs & Training](#) | [Online Support](#)
[Community Discussion](#) | [Industry News](#) | [Solutions Marketplace](#) | [Case Studies](#)

[Glossary](#) | [Feedback](#) | [A-Z Index](#)

For more information on Java technology
and other software from Sun Microsystems, call:
(800) 786-7638
Outside the U.S. and Canada, dial your country's
[AT&T Direct Access Number](#) first.



Copyright © 1995-2001 [Sun Microsystems, Inc.](#)
All Rights Reserved. [Terms of Use](#). [Privacy Policy](#).



[Products & APIs](#)
[Developer Connection](#)
[Docs & Training](#)
[Online Support](#)
[Community Discussion](#)
[Industry News](#)
[Solutions Marketplace](#)
[Case Studies](#)

[Printable Page](#) 

■ Requires login

[Early Access](#) ■
[Downloads](#)

[Bug Database](#) ■
[Submit a Bug](#)
[View Database](#)

[Newsletters](#)
[Back Issues](#)
[Subscribe](#)

[Learning Centers](#)
[Articles](#)
[Bookshelf](#)
[Code Samples](#)
[New to Java](#)
[Question of the Week](#)
[Quizzes](#)
[Tech Tips](#)
[Tutorials](#)

[Forums](#)

[Training Index](#)

Writing Advanced Applications

Chapter 6: Project Swing: Building a User Interface

[<<[BACK](#)] [[CONTENTS](#)] [[NEXT](#)>>]

The Java™ Foundation Classes (JFC) Project Swing and Enterprise JavaBeans™ architectures share one key design element: the separation of data from the display or manipulation of that data. In Enterprise JavaBeans applications, the entity bean provides a view of the data. The underlying data storage mechanism can be swapped out and replaced without changing the entity bean view or recompiling any code that uses the view.

Project Swing separates the view and control of a visual component from its contents, or data model. However, although Project Swing does have the components that make up a Model-View-Controller (MVC) architecture, it is more accurately described as a model-delegate architecture. This is because the controller part of the Project Swing interface, often the mouse and keyboard events the component responds to, is combined with the physical view in one User Interface delegate (UI delegate) object.

Each component, for example a `JButton` or a `JScrollBar`, has a separate UI delegate class that inherits from the `ComponentUI` class and is under the control of a separate UI manager. While each component has a basic UI delegate, it is no longer tied to the underlying data so a new set of delegates -- a set of metal-styled components, for example -- can be swapped in while the application is still running. The ability to change the look and behavior reflects the pluggable look and feel (PLAF) feature available in Project Swing.

This chapter describes Project Swing user interface components in terms of the `AuctionClient` example application.

[Components and Data Models](#)

[Printing API](#)

[Advanced Printing](#)

Technology Centers

In a Rush?

This table links you directly to specific topics.

Topic	Section
Components and Data Models	Lightweight Components Ordering Components Data Models Custom Cell Rendering Custom Cell Editing Specialized Event Handling Project Swing Directions

[\[TOP\]](#)

Printable Page 

[This page was updated: 4-Jun-2001]

[Products & APIs](#) | [Developer Connection](#) | [Docs & Training](#) | [Online Support](#)
[Community Discussion](#) | [Industry News](#) | [Solutions Marketplace](#) | [Case Studies](#)

[Glossary](#) | [Feedback](#) | [A-Z Index](#)

For more information on Java technology and other software from Sun Microsystems, call: (800) 786-7638
Outside the U.S. and Canada, dial your country's [AT&T Direct Access Number](#) first.



Copyright © 1995-2001 [Sun Microsystems, Inc.](#)
All Rights Reserved. [Terms of Use](#). [Privacy Policy](#).



[Products & APIs](#)
[Developer Connection](#)
[Docs & Training](#)
[Online Support](#)
[Community Discussion](#)
[Industry News](#)
[Solutions Marketplace](#)
[Case Studies](#)

[Training Index](#)

Writing Advanced Applications

Chapter 6 Continued: Components and Data Models

[<<BACK] [CONTENTS] [NEXT>>]

[Printable Page](#)

■ Requires login

[Early Access](#) ■
[Downloads](#)

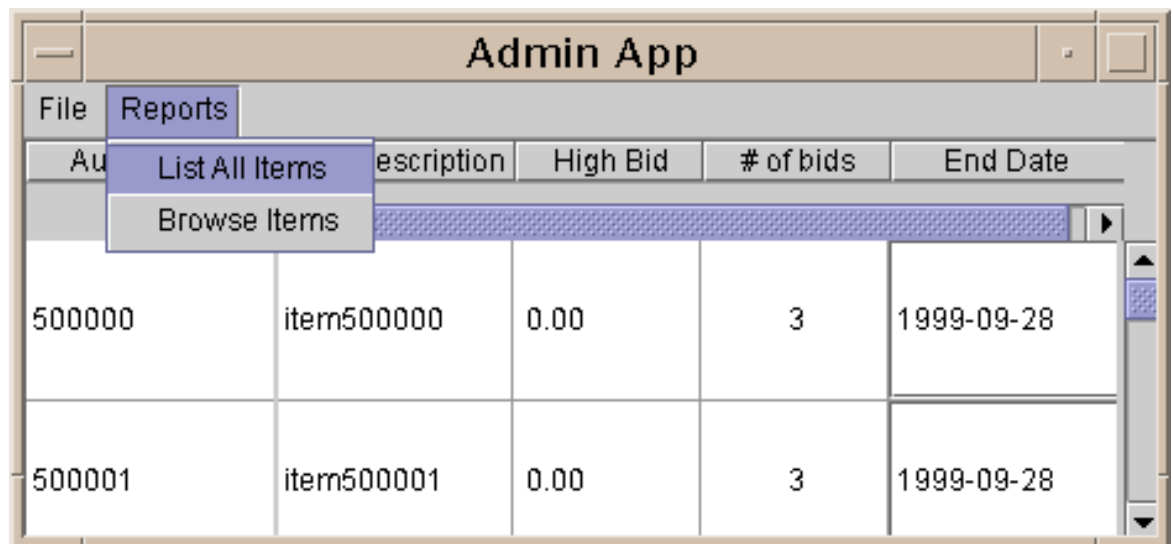
[Bug Database](#) ■
[Submit a Bug](#)
[View Database](#)

[Newsletters](#)
[Back Issues](#)
[Subscribe](#)

[Learning Centers](#)
[Articles](#)
[Bookshelf](#)
[Code Samples](#)
[New to Java](#)
[Question of the Week](#)
[Quizzes](#)
[Tech Tips](#)
[Tutorials](#)

[Forums](#)

The [AuctionClient](#) program is a simple GUI application that lets auction administrators list and browse auction items, and print auction item reports. This section describes the Project Swing application code, which uses lightweight components and the other Project Swing features shown in the bullet list.



[Lightweight Components](#)
[Ordering Components](#)
[Data Models](#)
[Custom Cell Rendering](#)
[Custom Cell Editing](#)
[Specialized Event Handling](#)
[Project Swing Directions](#)

Lightweight Components

All components in Project Swing, except `JApplet`, `JDialog`, `JFrame` and `JWindow` are lightweight components. Lightweight components,

Technology Centers unlike their Abstract Window Toolkit (AWT) counterparts, do not depend on the local windowing toolkit.

For example, a heavyweight `java.awt.Button` running on the Java™ platform for the Unix platform maps to a real Motif button. In this relationship, the Motif button is called the peer to the `java.awt.Button`. If you create two `java.awt.Button` in an application, two peers and hence two Motif Buttons are also created. The Java platform communicates with the Motif Buttons using the Java Native Interface. For each and every component added to the application, there is an additional overhead tied to the local windowing system, which is why these components are called heavyweight.

Lightweight components are termed peerless components and emulate the local window system components. A lightweight button is represented as a rectangle with a label inside that accepts mouse events. Adding more lightweight buttons means drawing more rectangles.

A lightweight component needs to be drawn on something, and an application written in the Java programming language needs to interact with the local window manager so the main application window can be closed or minimized. This is why the top-level parent components mentioned above (`JFrame`, `JApplet`, and others) are implemented as heavyweight components -- they need to be mapped to a component in the local window toolkit.

A `JButton` is a very simple shape to draw. For more complex components like `JList` or `JTable`, the elements or cells of the list or table are drawn by a `CellRenderer` object. A `CellRenderer` object provides flexibility because it makes it possible for any type of object to be displayed in any row or column.

For example, a `JTable` can use a different `CellRenderer` for each column. This code segment sets the second column, which is referenced as index 1, to use a `CustomRenderer` object to create the cells for that column.

```
JTable scrollTable=new JTable(rm);
TableColumnModel scrollColumnModel =
    scrollTable.getColumnModel();
CustomRenderer custom = new CustomRenderer();
scrollColumnModel.getColumnModel(1).setCellRenderer(custom);
```

Ordering Components

Each Project Swing applet or application needs at least one

heavyweight container component (a `JFrame`, `JWindow`, `JApplet`, or `JDialog`). Each of these containers with `JFrame`'s lightweight MDI counterpart, `JInternalFrame`, contains a component called a `RootPane`. The `JRootPane` manages the additional layers used in the container such as the `JLayeredPane`, `JContentPane`, `GlassPane` and the optional `JMenuBar`. It also lets all emulated (lightweight) components interact with the AWT event queue to send and receive events. Interacting with the event queue gives emulated components indirect interaction with the local window manager.

`JLayeredPane`

The `JLayeredPane` sits on top of the `JRootPane`, and as its name implies, controls the layers of the components contained within the boundary of the heavyweight container. The components are not added to the `JLayeredPane`, but to the `JContentPane` instead. The `JLayeredPane` determines the Z-ordering of the components in the `JRootPane`. The Z-order can be thought of as the order of overlay among various the components. If you drag-and-drop a component or request a dialog to popup, you want that component to appear in front of the others in the application window. The `JLayeredPane` lets you layer components.

The `JLayeredPane` divides the depth of the container into different bands that can be used to assign a component to a type-appropriate level. The `DRAG_LAYER` band, value 400, appears above all other defined component layers. The lowermost level of `JLayeredpane`, the `DEFAULT_FRAME_LAYER` band, has value -3000 and is the level of the heavyweight containers, including the `MenuBar`. The bands are as follows:

Value	Layer Name	Component Types
-3000	<code>DEFAULT_FRAME_LAYER</code>	<code>JMenuBar</code>
0	<code>DEFAULT_LAYER</code>	<code>JButton</code> , <code>JTable</code> , ..
	<code>PALETTE_LAYER</code>	Floating components such as <code>JToolBar</code>
	<code>MODAL_LAYER</code>	Modal Dialogs
400	<code>DRAG_LAYER</code>	Drag-and-drop over all layers



Within these general depth bands, components can be further arranged with another numbering system to order the components in a particular band, but this system reverses the numbering priority. For example, in a specific band such as `DEFAULT_LAYER`, components with a value of 0 appear in front of others in that band; whereas, components with a higher number or -1 appear behind them. The highest number in this scheme is the number of components minus 1, so one way to visualize it is a vector of components that steps through painting the components with a higher number first finishing with the one at position 0.

For example, the following code adds a `JButton` to the default layer and specifies that it appear in front of the other components in that same layer:

```

        JButton enterButton = new JButton("Enter");
        layeredPane.add(enterButton,
            JLayeredPane.Default_Layer, 0);
    
```

You can achieve the same effect by calling the `LayeredPane.moveToFront` method within a layer or using the `LayeredPane.setLayer` method to move to a different layer.

JContentPane

The `JContentPane` manages adding components to heavyweight containers. So, you have to call the `getContentPane` method to add a component to the `ContentPane` of the `RootPane`. By default, a `ContentPane` is initialized with a `BorderLayout` layout manager. There are two ways to change the layout manager. You can call the `setLayout` method like this:

```

    getContentPane().setLayout(new BorderLayout());
    
```

Or you can replace the default `ContentPane` with your own `ContentPane`, such as a `JPanel`, like this:

```

    JPanel pane= new JPanel();
    pane.setLayout(new BorderLayout());
    setContentPane(pane);
    
```

GlassPane

The `GlassPane` is usually completely transparent and just acts as a sheet of glass in front of the components. You can implement your own `GlassPane` by using a component like `JPanel` and installing it as the `GlassPane` by calling the `setGlassPane` method. The `RootPane` is configured with a `GlassPane` that can be retrieved by calling `getGlassPane`.

One way to use a `GlassPane` is to implement a component that invisibly handles all mouse and keyboard events, effectively blocking user input until an event completes. The `GlassPane` can block the events, but currently the cursor will not return to its default state if you have set the cursor to be a busy cursor in the `GlassPane`. An additional mouse event is required for the refresh.

```
MyGlassPane glassPane = new MyGlassPane();
setGlassPane(glassPane);
setGlassPane.setVisible(true); //before worker thread
..
setGlassPane.setVisible(false); //after worker thread

private class MyGlassPane extends JPanel {

    public MyGlassPane() {
        addKeyListener(new KeyAdapter() { });
        addMouseListener(new MouseAdapter() { });
        super.setCursor(
            Cursor.getPredefinedCursor(Cursor.WAIT_CURSOR));
    }
}
```

Data Models

Numerous model layers are combined to form the tables of the `AuctionClient` GUI. At a foundational level, the `TableModel` interface and its two implementations `AbstractTableModel` and `DefaultTableModel` provide the most basic means for storage, retrieval and modification of the underlying data.

The `TableModel` is responsible for defining and categorizing the data by its class. It also determines if the data can be edited and how the data is grouped into columns and rows. It is important to note, however, that while the `TableModel` interface is used most often in the construction of a `JTable`, it is not fundamentally tied to their display. Implementations could just as easily form the basis of a spreadsheet component, or even a non-GUI class that calls for the organization of data in tabular format.

The `ResultsModel` class is at the heart of the `AuctionClient` tables. It defines a dynamic data set, dictates whether class users can edit the data through its `ResultsModel.isCellEditable` method, and provides the `update` method to keep the data current. The model underlies the scrolling and fixed tables, and lets modifications to be reflected in each view.

At a higher level, and representing an intermediate layer between data and its graphical representation, is the `TableColumnModel`. At this level the data is grouped by column in anticipation of its

ultimate display in the table. The visibility and size of these columns, their headers, and the component types of their cell renderers and editors are all managed by the `TableColumnModel` class.

For example, freezing the left-most columns in the `AuctionClient` GUI is possible because column data is easily exchanged among multiple `TableColumnModel` and `JTable` objects. This translates to the `fixedTable` and `scrollTable` objects of the `AuctionClient` program.

Higher still lie the various renderers, editors, and header components whose combination define the look and organization of the `JTable` component. This level is where the fundamental layout and display decisions of the `JTable` are made.

The creation of the inner classes `CustomRenderer` and `CustomButtonRenderer` within the `AuctionClient` application allows users of those classes to redefine the components upon which the appearance of table cells are based. Likewise, the `CustomButtonEditor` class takes the place of the table's default editor. In true object-oriented fashion, the default editors and renderers are easily replaced, affecting neither the data they represent nor the function of the component in which they reside.

Finally, the various component user interfaces are responsible for the ultimate appearance of the `JTable`. It is here the look-and-feel-specific representation of the `AuctionClient` tables and their data are rendered in final form to the user. The end result is that adding a Project Swing front-end to existing services requires little additional code. In fact, coding the model is one of the easier tasks in building a Project Swing application.

Table Model

The `JTable` class has an associated `DefaultTableModel` class that internally uses a `Vector` of vectors to store data internally. The data for each row is stored in a single `Vector` object while another `Vector` object stores each of those rows as its constituent elements. The `DefaultTableModel` object can be initialized with data in several different ways. This code shows the `DefaultTableModel` created with a two-dimensional array and a second array representing column headings. The `DefaultTableModel` in turn converts the `Object` arrays into the appropriate `Vector` objects:

```
Object[][] data = new Object[][]{ {"row 1 col1",
                                   "Row 1 col2" },
                                   {"row 2 col 1",
                                   "row 2 col 2"}}
```

```

    };
    Object[] headers = new Object[] {"first header",
                                     "second header"};
    DefaultTableModel model = new DefaultTableModel(data,
                                                    headers);

    table = new JTable(model);
    table.setAutoResizeMode(JTable.AUTO_RESIZE_OFF);

```

Creating a custom table model is nearly as easy as using `DefaultTableModel`, and requires little additional coding. You can implement a table model by implementing a method to return the number of entries in the model, and a method to retrieve an element at a specific position in that model. For example, the `JTable` model can be implemented from

`javax.swing.table.AbstractTableModel` by implementing the methods `getColumnCount`, `getRowCount` and `getValueAt` as shown here:

```

    final Object[][] data = new Object[][]{ {
        "row 1 col1",
        "row 1 col2" },
        {"row 2 col 1",
        "row 2 col 2"} };
    final Object[] headers = new Object[] {
        "first header",
        "second header"};

    TableModel model = new AbstractTableModel(){
        public int getColumnCount() {
            return data[0].length;
        }
        public int getRowCount() {
            return data.length;
        }
        public String getColumnName(int col) {
            return (String)headers[col];
        }

        public Object getValueAt(int row,int col) {
            return data[row][col];
        }
    };
    table = new JTable(model);
    table.setAutoResizeMode(
        JTable.AUTO_RESIZE_OFF);

```

This table is read-only and its data values are already known. In fact, the data is even declared `final` so it can be retrieved by the inner `TableModel` class. This is not normally the situation when working with live data.

You can create an editable table by adding the `isCellEditable` verification method, which is used by the default cell editor, and the `AbstractTableModel` method for setting a value at a position. Up until this change, the `AbstractTableModel` has been handling the repainting and resizing of the table by firing different table changed events. Because the `AbstractTableModel` does not know that

something has occurred to the table data, you need to inform it by calling the `fireTableCellUpdated` method. The following lines are added to the `AbstractTableModel` inner class to allow editing of the data:

```
public void setValueAt (Object value,
                       int row, int col) {
    data[row][col] = value;
    fireTableCellUpdated (row, col);
}

public boolean isCellEditable(int row,
                              int col) {
    return true;
}
```

More Table Models

A common requirement for the display of tabular data is the inclusion of a non-scrolling column. This column provides a set of anchor data that remains stationary and visible while its neighboring columns are scrolled horizontally (and often out of view). This is particularly important in cases where row data can be identified by a unique value in the fixed column, such as a name or identification number. The next code example uses a fixed table column to display a list of the auction items.

The base table model in this example implements the `AbstractTableModel` class. Its `update` method dynamically populates the table data from a call to the database. It sends an event that the table has been updated by calling the `fireTableStructureChanged` method to indicate the number of rows or columns in the table have changed.

```
package auction;

import javax.swing.table.AbstractTableModel;
import javax.swing.event.TableModelEvent;
import java.text.NumberFormat;
import java.util.*;
import java.awt.*;

public class ResultsModel extends AbstractTableModel{
    String[] columnNames={};
    Vector rows = new Vector();

    public String getColumnName(int column) {
        if (columnNames[column] != null) {
            return columnNames[column];
        } else {
            return "";
        }
    }

    public boolean isCellEditable(int row, int column){
```

```

        return false;
    }

    public int getColumnCount() {
        return columnNames.length;
    }

    public int getRowCount() {
        return rows.size();
    }

    public Object getValueAt(int row, int column){
        Vector tmprow = (Vector)rows.elementAt(row);
        return tmprow.elementAt(column);
    }

    public void update(Enumeration enum) {
        try {
            columnNames = new String[5];
            columnNames[0]=new String("Auction Id #");
            columnNames[1]=new String("Description");
            columnNames[2]=new String("High Bid");
            columnNames[3]=new String("# of bids");
            columnNames[4]=new String("End Date");
            while((enum !=null) &&
                (enum.hasMoreElements())) {
                while(enum.hasMoreElements()) {
                    AuctionItem auctionItem=(
                    AuctionItem)enum.nextElement();
                    Vector items=new Vector();
                    items.addElement(new Integer(
                    auctionItem.getId()));
                    items.addElement(
                    auctionItem.getSummary());
                    int bidcount= auctionItem.getBidCount();
                    if(bidcount >0) {
                        items.addElement(
                        NumberFormat.getCurrencyInstance().
                        format(auctionItem.getHighBid()));
                    } else {
                        items.addElement("-");
                    }
                    items.addElement(new Integer(bidcount));
                    items.addElement(auctionItem.getEndDate());
                    rows.addElement(items);
                }
            }

            fireTableStructureChanged();
        } catch (Exception e) {
            System.out.println("Exception e"+e);
        }
    }
}

```

The table is created from the `ResultsModel` model. Then, the first table column is removed from that table and added to a new table. Because there are now two tables, the only way the selections can be kept in sync is to use a `ListSelectionModel` object to set the selection on the table row in the other tables that were not

selected by calling the `setRowSelectionInterval` method.

The full example can be found in the [AuctionClient.java](#) source file:

```
private void listAllItems() throws IOException{
    ResultsModel rm=new ResultsModel();
    if (!standaloneMode) {
        try {
            BidderHome bhome=(BidderHome)
            ctx.lookup("bidder");
            Bidder bid=bhome.create();
            Enumeration enum=
            (Enumeration)bid.getItemList();
            if (enum != null) {
                rm.update(enum);
            }
        } catch (Exception e) {
            System.out.println(
            "AuctionServlet <list>:"+e);
        }
    } else {
        TestData td= new TestData();
        rm.update(td.results());
    }
    scrollTable=new JTable(rm);
    adjustColumnWidth(scrollTable.getColumnModel(
        "End Date"), 150);
    adjustColumnWidth(scrollTable.getColumnModel(
        "Description"), 120);
    scrollColumnModel = scrollTable.getColumnModel();
    fixedColumnModel = new DefaultTableColumnModel();

    TableColumn col = scrollColumnModel.getColumnModel(0);
    scrollColumnModel.removeColumn(col);
    fixedColumnModel.addColumn(col);

    fixedTable = new JTable(rm,fixedColumnModel);
    fixedTable.setRowHeight(scrollTable.getRowHeight());
    headers = new JViewport();

    ListSelectionModel fixedSelection =
        fixedTable.getSelectionModel();
    fixedSelection.addListSelectionListener(
        new ListSelectionListener() {
            public void valueChanged(ListSelectionEvent e) {
                ListSelectionModel lsm = (
                    ListSelectionModel)e.getSource();
                if (!lsm.isSelectionEmpty()) {
                    setScrollableRow();
                }
            }
        });

    ListSelectionModel scrollSelection =
        scrollTable.getSelectionModel();
    scrollSelection.addListSelectionListener(
        new ListSelectionListener() {
            public void valueChanged(ListSelectionEvent e) {
                ListSelectionModel lsm =
```

```

        (ListSelectionModel)e.getSource();
        if (!lsm.isSelectionEmpty()) {
            setFixedRow();
        }
    }
});

CustomRenderer custom = new CustomRenderer();
custom.setHorizontalAlignment(JLabel.CENTER);
scrollColumnModel.getColumn(2).setCellRenderer(
    custom);
scrollColumnModel.getColumn(3).setCellRenderer(
    new CustomButtonRenderer());

CustomButtonEditor customEdit=new
    CustomButtonEditor(frame);
scrollColumnModel.getColumn(3).setCellEditor(
    customEdit);

headers.add(scrollTable.getTableHeader());

JPanel topPanel = new JPanel();
topPanel.setLayout(new BorderLayout(topPanel,
    BorderLayout.X_AXIS));
adjustColumnWidth(
    fixedColumnModel.getColumn(0), 100);

JTableHeader fixedHeader=
    fixedTable.getTableHeader();
fixedHeader.setAlignmentY(Component.TOP_ALIGNMENT);
topPanel.add(fixedHeader);
topPanel.add(Box.createRigidArea(
    new Dimension(2, 0)));
topPanel.setPreferredSize(new Dimension(400, 40));

JPanel headerPanel = new JPanel();
headerPanel.setAlignmentY(Component.TOP_ALIGNMENT);
headerPanel.setLayout(new BorderLayout());

JScrollPane scrollpane = new JScrollPane();
scrollBar = scrollpane.getHorizontalScrollBar();

headerPanel.add(headers, "North");
headerPanel.add(scrollBar, "South");
topPanel.add(headerPanel);

scrollTable.setPreferredSize(
    new Dimension(300,180));
fixedTable.setPreferredSize(
    new Dimension(100,180));
fixedTable.setPreferredSize(
    new Dimension(100,180));

innerPort = new JViewport();
innerPort.setView(scrollTable);
scrollpane.setViewPort(innerPort);

scrollBar.getModel().addChangeListener(
    new ChangeListener() {
        public void stateChanged(ChangeEvent e) {
            Point q = headers.getViewPosition();

```



```

        Point p = innerPort.getViewPosition();
        int val = scrollBar.getModel().getValue();
        p.x = val;
        q.x = val;
        headers.setViewPosition(p);
        headers.repaint(headers.getViewRect());
        innerPort.setViewPosition(p);
        innerPort.repaint(innerPort.getViewRect());
    }
});

scrollTable.getTableHeader(
    ).setUpdateTableInRealTime(
        false);

JPanel bottomPanel = new JPanel();
bottomPanel.setLayout(new BorderLayout(
    bottomPanel, BorderLayout.X_AXIS));
fixedTable.setAlignmentY(Component.TOP_ALIGNMENT);
bottomPanel.add(fixedTable);
bottomPanel.add(Box.createRigidArea(
    new Dimension(2, 0)));
innerPort.setAlignmentY(Component.TOP_ALIGNMENT);
bottomPanel.add(innerPort);
bottomPanel.add(Box.createRigidArea(
    new Dimension(2, 0)));

scrollPane= new JScrollPane(bottomPanel,
    JScrollPane.VERTICAL_SCROLLBAR_ALWAYS,
    JScrollPane.HORIZONTAL_SCROLLBAR_NEVER);
JViewport outerPort = new JViewport();
outerPort.add(bottomPanel);
scrollPane.setColumnHeaderView(topPanel);
scrollPane.setViewPort(outerPort);

scrollTable.setAutoResizeMode(
    JTable.AUTO_RESIZE_OFF);
frame.getContentPane().add(scrollPane);

scrollTable.validate();
frame.setSize(450,200);
}

void setFixedRow() {
    int index=scrollTable.getSelectedRow();
    fixedTable.setRowSelectionInterval(index, index);
}

void setScrollableRow() {
    int index=fixedTable.getSelectedRow();
    scrollTable.setRowSelectionInterval(index, index);
}

void adjustColumnWidth(TableColumn c, int size) {
    c.setPreferredWidth(size);
    c.setMaxWidth(size);
    c.setMinWidth(size);
}
}

```

JList Model

The `JList` component displays a vertical list of data elements and uses a `ListModel` to hold and manipulate the data. It also uses a `ListSelectionModel` object to enable selection and subsequent retrieval of elements in the list.

Default implementations of the `AbstractListModel` and `AbstractListSelectionModel` classes are provided in the Project Swing API in the form of the `DefaultListModel` and `DefaultListSelectionModel` classes. If you use these two default models and the default cell renderer, you get a list that displays model elements by calling the `toString` method on each object. The list uses the `MULTIPLE_INTERVAL_SELECTION` list selection model to select each element from the list.

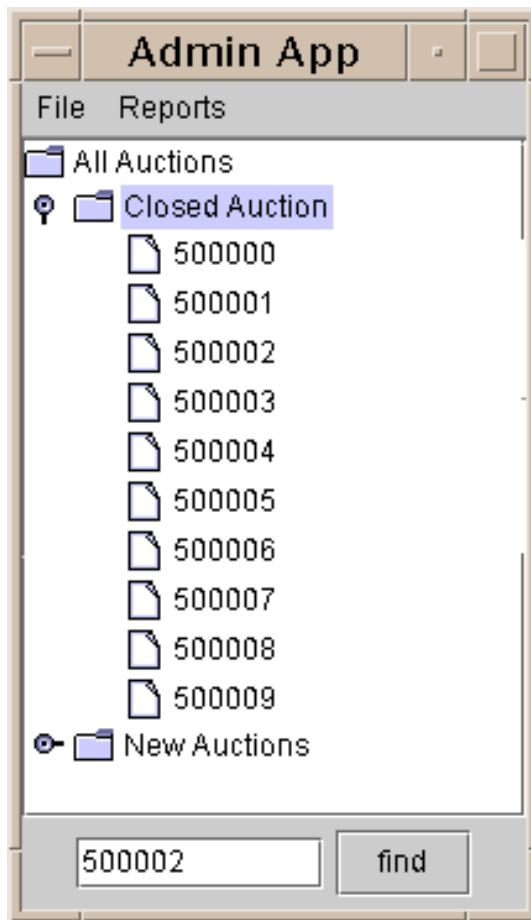
Three selection modes are available to `DefaultListSelectionModel`: `SINGLE_SELECTION`, where only one item is selected at a time; `SINGLE_INTERVAL_SELECTION` in which a range of sequential items can be selected; and `MULTIPLE_INTERVAL_SELECTION`, which allows any or all elements to be selected. The selection mode can be changed by calling the `setSelectionMode` method in the `JList` class.

```
public SimpleList() {
    JList list;
    DefaultListModel deflist;
    deflist= new DefaultListModel();
    deflist.addElement("element 1");
    deflist.addElement("element 2");
    list = new JList(deflist);

    JScrollPane scroll = new JScrollPane(list);
    getContentPane().add(scroll, BorderLayout.CENTER);
}
```

JTree Model

The `JTree` class models and displays a vertical list of elements or nodes arranged in a tree-based hierarchy.



A `JTree` object has one root node and one or more child nodes, which can contain further child nodes. Each parent node can be expanded to show all its children similar to directory trees familiar to Windows users.

Like the `JList` and `JTable` components, the `JTree` consists of more than one model. The selection model is similar to the one detailed for the `JList` model. The selection modes have the following slightly different names:

`SINGLE_TREE_SELECTION`,
`DISCONTIGUOUS_TREE_SELECTION`, and
`CONTIGUOUS_TREE_SELECTION`.

While `DefaultTreeModel` maintains the data in the tree and is responsible for adding and removing nodes, it is the `DefaultMutableTreeNode` class that defines the methods used for node traversal. The `DefaultTreeModel` is often used to implement custom models because there is no `AbstractTreeModel` in the `JTree` package. However, if you use custom objects, you must implement `TreeModel`. This code example creates a `JTree` using the `DefaultTreeModel`.

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.tree.*;

public class SimpleTree extends JFrame {
    public SimpleTree() {
        String[] treelabels = {
            "All Auctions",
            "Closed Auction",
            "Open Auctions"};

        Integer[] closedItems = { new Integer(500144),
            new Integer(500146),
            new Integer(500147)};

        Integer[] openItems = { new Integer(500148),
            new Integer(500149)};

        DefaultMutableTreeNode[] nodes = new
            DefaultMutableTreeNode[treelabels.length];
        DefaultMutableTreeNode[] closednodes = new
            DefaultMutableTreeNode[closedItems.length];
        DefaultMutableTreeNode[] opennodes = new
```

```

        DefaultMutableTreeNode[openItems.length];

    for (int i=0; i < treelabels.length; i++) {
        nodes[i] = new
            DefaultMutableTreeNode(treelabels[i]);
    }
    nodes[0].add(nodes[1]);
    nodes[0].add(nodes[2]);

    for (int i=0; i < closedItems.length; i++) {
        closednodes[i] = new
            DefaultMutableTreeNode(closedItems[i]);
        nodes[1].add(closednodes[i]);
    }

    for (int i=0; i < openItems.length; i++) {
        opennodes[i] = new
            DefaultMutableTreeNode(openItems[i]);
        nodes[2].add(opennodes[i]);
    }
    DefaultTreeModel model=new
        DefaultTreeModel(nodes[0]);

    JTree tree = new JTree(model);

    JScrollPane scroll = new JScrollPane(tree);
    getContentPane().add(scroll, BorderLayout.CENTER);
}

public static void main(String[] args) {
    SimpleTree frame = new SimpleTree();
    frame.addWindowListener( new WindowAdapter() {
        public void windowClosing( WindowEvent e ) {
            System.exit(0);
        }
    });
    frame.setVisible(true);
    frame.pack();
    frame.setSize(150,150);
}
}

```

The `toString` method is used to retrieve the value for the `Integer` objects in the tree. And although the `DefaultTreeModel` is used to maintain the data in the tree and to add or remove nodes, the `DefaultMutableTreeNode` class defines the methods used to traverse through the nodes in the tree.

A primitive search of the nodes in a `JTree` is accomplished with the `depthFirstEnumeration` method, which is the same as the `postorderEnumeration` method and works its way from the end points of the tree first. Or you can call the `preorderEnumeration` method, the reverse of the `postorderEnumeration` method, which starts from the root and descends each tree in turn. Or you can call the `breadthFirstEnumeration` method, which starts from the root and visits all the child nodes in one level before visiting the child nodes at a lower depth.

The following code expands the parent node if it contains a child node that matches the search field entered. It uses a call to `Enumeration e = nodes[0].depthFirstEnumeration();` to return a list of all the nodes in the tree. Once it has found a match, it builds the `TreePath` from the root node to the node that matched the search to pass to the `setVisible` method in the `JTree` class that ensures the node is expanded in the tree.

```
import java.awt.*;
import java.util.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.tree.*;

public class SimpleSearchTree extends JFrame {
    JPanel findPanel;
    JTextField findField;
    JTree tree;
    JButton findButton;
    DefaultMutableTreeNode[] nodes;

    public SimpleSearchTree() {
        String[] treelabels = { "All Auctions",
                                "Closed Auction",
                                "Open Auctions" };
        Integer[] closedItems = { new Integer(500144),
                                   new Integer(500146),
                                   new Integer(500147) };

        Integer[] openItems = { new Integer(500148),
                                   new Integer(500149) };

        nodes = new
            DefaultMutableTreeNode[treelabels.length];
        DefaultMutableTreeNode[] closednodes = new
            DefaultMutableTreeNode[closedItems.length];
        DefaultMutableTreeNode[] opennodes = new
            DefaultMutableTreeNode[openItems.length];
        for (int i=0; i < treelabels.length; i++) {
            nodes[i] = new
                DefaultMutableTreeNode(treelabels[i]);
        }
        nodes[0].add(nodes[1]);
        nodes[0].add(nodes[2]);

        for (int i=0; i < closedItems.length; i++) {
            closednodes[i] = new
                DefaultMutableTreeNode(closedItems[i]);
            nodes[1].add(closednodes[i]);
        }

        for (int i=0; i < openItems.length; i++) {
            opennodes[i] = new DefaultMutableTreeNode(
                openItems[i]);
            nodes[2].add(opennodes[i]);
        }

        DefaultTreeModel model=new
```

```

        DefaultTreeModel(nodes[0]);
tree = new JTree(model);

JScrollPane scroll = new JScrollPane(tree);
getContentPane().add(scroll, BorderLayout.CENTER);
findPanel= new JPanel();
findField= new JTextField(10);
findButton= new JButton("find");
findButton.addActionListener (new ActionListener() {
    public void actionPerformed (ActionEvent e) {
        String field=findField.getText();
        if (field != null) {
            findNode(findField.getText());
        } else {
            return;
        }
    }
});
findPanel.add(findField);
findPanel.add(findButton);
getContentPane().add(findPanel, BorderLayout.SOUTH);
}
public void findNode(String field) {
    Enumeration e = nodes[0].depthFirstEnumeration();
    Object currNode;
    while (e.hasMoreElements()) {
        currNode = e.nextElement();
        if (currNode.toString().equals(field)) {
            TreePath path=new TreePath(((
                DefaultMutableTreeNode)currNode).getPath());
            tree.makeVisible(path);
            tree.setSelectionRow(tree.getRowForPath(path));
            return;
        }
    }
}
}

public static void main(String[] args) {
    SimpleSearchTree frame = new SimpleSearchTree();
    frame.addWindowListener( new WindowAdapter() {
        public void windowClosing( WindowEvent e ) {
            System.exit(0);
        }
    });
    frame.setVisible(true);
    frame.pack();
    frame.setSize(300,150);
}
}

```

JTree, JTable and JList are probably the most common models you will want to customize. But you can use models such as SingleSelectionModel for general data manipulation. The SingleSelectionModel class lets you specify how data is selected in a component.

Custom Cell Rendering

As you learned above, many components have a default cell renderer to paint each element in a table, tree or list. The default cell renderer is usually a `JLabel` and displays a `String` representation of the data element.

A simple custom cell renderer can extend the `DefaultXXXCellRenderer` class to provide additional customization in the `getXXXCellRenderer`. The `DefaultTableCellRenderer` and `DefaultTreeCellRenderer` Components both use a `JLabel` to render the cell. This means any customization that can be applied to a `JLabel` can also be used in the `JTable` or `JTree` cell.

For example, the following renderer sets the background color of the component if the auction item has received a high number of bids:

```
class CustomRenderer extends DefaultTableCellRenderer {
    public Component getTableCellRendererComponent(
        JTable table, Object value,
        boolean isSelected,
        boolean hasFocus,
        int row, int column) {

        Component comp =
            super.getTableCellRendererComponent(
                table, value, isSelected, hasFocus,
                row, column);

        JLabel label = (JLabel)comp;

        if(((Integer)value).intValue() >= 30) {
            label.setIcon(new ImageIcon("Hot.gif"));
        } else {
            label.setIcon(new ImageIcon("Normal.gif"));
        }

        return label;
    }
}
```

The renderer is set on a column like this:

```
CustomRenderer custom = new CustomRenderer();
custom.setHorizontalAlignment(JLabel.CENTER);
scrollColumnModel.getColumn(2).setCellRenderer(
    custom);
```

If the component being displayed inside the `JTable` column requires more functionality than is available using a `JLabel`, you can create your own `TableCellRenderer`. This next code example uses a `JButton` as the renderer cell.

```
class CustomButtonRenderer extends JButton
    implements TableCellRenderer {
    public CustomButtonRenderer() {
        setOpaque(true);
    }
}
```

```

public Component getTableCellRendererComponent(
    JTable table, Object value,
    boolean isSelected,
    boolean hasFocus, int row,
    int column) {

    if (isSelected) {
        ((JButton)value).setForeground(
            table.getSelectionForeground());
        ((JButton)value).setBackground(
            table.getSelectionBackground());
    } else {
        ((JButton)value).setForeground(table.getForeground());
        ((JButton)value).setBackground(table.getBackground());
    }
    return (JButton)value;
}
}

```

Like the default `JLabel` cell renderer, this class relies on an underlying component (in this case `JButton`) to do the painting. Selection of the cell toggles the button colors. As before, the cell renderer is secured to the appropriate column of the auction table with the `setCellRenderer` method:

```

scrollColumnModel.getColumn(3).setCellRenderer(
    new CustomButtonRenderer());

```

Alternately, all `JButton` components can be configured to use the `CustomButtonRenderer` in the table with a call to `setDefaultRenderer` as follows:

```

table.setDefaultRenderer(
    JButton.class, new CustomButtonRenderer());

```

Custom Cell Editing

In the same way that you can configure how a cell is painted in a `JTable` or `JTree` component, you can also configure how an editable cell responds to edits. One difference between using cell editors and cell renderers is there is a `DefaultCellEditor` for all components, but no `DefaultTableCellEditor` for table cells.

While separate renderers exist for `JTree` and `JTable`, a single `DefaultCellEditor` class implements both the `TableCellEditor` and `TreeCellEditor` interfaces. However, the `DefaultCellEditor` class has constructors for only the `JComboBox`, `JCheckBox`, and `JTextField` components. The `JButton` class does not map to any of these constructors so a dummy `JCheckBox` is created to satisfy the requirements of the `DefaultCellEditor` class.

This next example uses a custom button editor that displays the number of days left in the auction when the button is double

clicked. The double click to trigger the action is specified by setting the value `clickCountToStart` to two. An exact copy of the `getTableCellEditorComponent` method paints the button in edit mode. A `JDialog` component that displays the number of days left appears when the `getCellEditorValue` method is called. The value for the number of days left is calculated by moving the current calendar date towards the end date. The `Calendar` class does not have a method that expresses a difference in two dates in anything other than the milliseconds between those two dates.

```
class CustomButtonEditor extends DefaultCellEditor {
    final JButton mybutton;
    JFrame frame;

    CustomButtonEditor(JFrame frame) {
        super(new JCheckBox());
        mybutton = new JButton();
        this.editorComponent = mybutton;
        this.clickCountToStart = 2;
        this.frame=frame;
        mybutton.setOpaque(true);
        mybutton.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                fireEditingStopped();
            }
        });
    }

    protected void fireEditingStopped() {
        super.fireEditingStopped();
    }

    public Object getCellEditorValue() {
        JDialog jd= new JDialog(frame, "Time left");
        Calendar today=Calendar.getInstance();
        Calendar end=Calendar.getInstance();
        SimpleDateFormat in=new SimpleDateFormat("yyyy-MM-dd");
        try {
            end.setTime(in.parse(mybutton.getText()));
        } catch (Exception e){
            System.out.println("Error in date"+mybutton.getText()+e);
        }
        int days = 0;
        while(today.before(end)) {
            today.roll(Calendar.DATE,true);
            days++;
        }
        jd.setSize(200,100);
        if (today.after(end)) {
            jd.getContentPane().add(new JLabel("Auction completed"));
        } else {
            jd.getContentPane().add(new JLabel("Days left="+days));
        }
        jd.setVisible(true);
        return new String(mybutton.getText());
    }

    public Component getTableCellEditorComponent(JTable table,
```

```

        Object value, boolean isSelected,
        int row, int column) {

    ((JButton) editorComponent).setText(((
        JButton)value).getText());
    if (isSelected) {
        ((JButton) editorComponent).setForeground(
            table.getSelectionForeground());
        ((JButton) editorComponent).setBackground(
            table.getSelectionBackground());
    } else {
        ((JButton) editorComponent).setForeground(
            table.getForeground());
        ((JButton) editorComponent).setBackground(
            table.getBackground());
    }
    return editorComponent;
}
}

```

Specialized Event Handling

Project Swing uses the event handling classes available in the AWT API since JDK 1.1. However, some new APIs are available in the `SwingUtilities` class that are used to add some control over the event queue. The two new event handling methods are `invokeLater` and `invokeAndWait`. The `invokeAndWait` method waits for the event to be processed in the event queue.

These methods are often used to request focus on a component after another event has occurred that might affect the component focus. You can return the focus by calling the `invokeLater` method and passing a `Thread`:

```

JButton button =new JButton();
SwingUtilities.invokeLater(new Runnable() {
    public void run() {
        button.requestFocus();
    }
});

```


Project Swing Directions

While the basic architecture of Project Swing has stayed true to its original design, many optimizations and improvements have been made to components like `JTable` and in areas such as scrolling. Add to this the Java HotSpot™ Performance Engine, which greatly reduces the cost of object creation, and Project Swing can boast its best performance to date.

However, as seen in the [Analyze a Program](#) section in the Performance chapter, a simple 700x300 table requires nearly half

a megabyte of memory when double buffered. The creation of ten tables would probably require swapping memory to disk, severely affecting performance on low-end machines.

[\[TOP\]](#)

Printable Page 

[This page was updated: 4-Jun-2001]

[Products & APIs](#) | [Developer Connection](#) | [Docs & Training](#) | [Online Support](#)
[Community Discussion](#) | [Industry News](#) | [Solutions Marketplace](#) | [Case Studies](#)

[Glossary](#) | [Feedback](#) | [A-Z Index](#)

For more information on Java technology
and other software from Sun Microsystems, call:
(800) 786-7638
Outside the U.S. and Canada, dial your country's
[AT&T Direct Access Number](#) first.



Copyright © 1995-2001 [Sun Microsystems, Inc.](#)
All Rights Reserved. [Terms of Use](#). [Privacy Policy](#).



Products & APIs
 Developer Connection
 Docs & Training
 Online Support
 Community Discussion
 Industry News
 Solutions Marketplace
 Case Studies

[Training Index](#)

Printable Page 

■ Requires login

Early Access ■
 Downloads

Bug Database ■
 Submit a Bug
 View Database

Newsletters
 Back Issues
 Subscribe

Learning Centers
 Articles
 Bookshelf
 Code Samples
 New to Java
 Question of the Week
 Quizzes
 Tech Tips
 Tutorials

Forums

Writing Advanced Applications

Chapter 6 Continued: Printing API

[\[<<BACK\]](#) [\[CONTENTS\]](#) [\[NEXT>>\]](#)

The Java™ 2 platform `java.awt.print` package lets you print anything that can be rendered to a `Graphics` or `Graphics2D` context — including AWT components, Java Foundation Classes (JFC) Project Swing (Project Swing) components, and 2D graphics. The Printing API is easy to use. Your application tells the printing system what to print, and the printing system determines when each page is rendered. This *callback printing model* enables printing support on a wide range of printers and systems. The callback model also lets users print to a bitmap printer from a computer that does not have enough memory or disk space to hold the bitmap for an entire page.

A graphics context lets a program paint to a rendering device such as a screen, printer, or offscreen image. Because Swing components are rendered through a `Graphics` object using AWT graphics support, it is easy to print Swing components with the new printing API. However, AWT components are not rendered to a graphics device, so you must extend the AWT component class and implement the AWT component paint method.

[What is in the Package?](#)

[Printing an AWT Component](#)

[Printing a Project Swing Component](#)

[Printing Graphics in Project Swing](#)

[Print Dialog](#)

[Page Setup Dialog](#)

[Printing a Collection of Pages](#)

What is in the Package?

The `java.awt.print` consists of the following interfaces, classes, and exceptions. Here is where you can view the [API specification](#).

Technology Centers

Interfaces

- Pageable
- Printable
- PrinterGraphics

Classes

- Book
- PageFormat
- Paper
- PrinterJob

Exceptions

- PrinterAbortException
- PrinterException
- PrinterIOException

Printing an AWT Component

`MyButton` The [printbutton.java](#) application displays a panel with *MyButton* on it. When you click the button, the application prints the *MyButton* component.

In the code, the `Button` class is extended to implement `Printable` and includes the `paint` and `print` method implementations. The `print` method is required because the class implements `Printable`, and the `paint` method is needed to describe how the button shape and label text looks when printed.

To see the button, the printer graphics context is translated into the imageable area of the printer, and to see the label text, a font is set on the printer graphics context.

In this example, the button is printed at a 164/72 inches inset from the left imageable margin (there are 72 pixels per inch) and 5/72 inches from the top imageable margin. This is where the button is positioned in the frame by the layout manager and those same numbers are returned by the following calls:

```
int X = (int)this.getLocation().getX();
int Y = (int)this.getLocation().getY();
```

And here is the `MyButton` class code:

```
class MyButton extends Button
    implements Printable {

    public MyButton() {
        super("MyButton");
    }
}
```

```

public void paint(Graphics g) {
//To see the label text, you must specify a font for
//the printer graphics context
    Font f = new Font("Monospaced", Font.PLAIN,12);
    g2.setFont (f);

//Using "g" render anything you want.
//Get the button's location, width, and height
    int X = (int)this.getLocation().getX();
    int Y = (int)this.getLocation().getY();
    int W = (int)this.getSize().getWidth();
    int H = (int)this.getSize().getHeight();

//Draw the button shape
    g.drawRect(X, Y, W, H);

//Draw the button label
//For simplicity code to center the label inside the
//button shape is replaced by integer offset values
    g.drawString(this.getLabel(), X+10, Y+15);
}

public int print(Graphics g,
                PageFormat pf, int pi)
    throws PrinterException {
    if (pi >= 1) {
        return Printable.NO_SUCH_PAGE;
    }

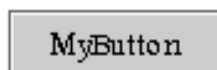
    Graphics2D g2 = (Graphics2D) g;

//To see the button on the printed page, you
//must translate the printer graphics context
//into the imageable area
    g2.translate(pf.getImageableX(), pf.getImageableY());
    g2.setColor(Color.black);
    paint(g2);
    return Printable.PAGE_EXISTS;
}

```

Note: The printing `Graphics2D` is based on the `BufferedImage` class and on some platforms does not default to a foreground color of black. If this is the case on your platform, you have to add `g2.setColor(Color.black)` to the `print` method before the `paint` invocation.

Printing a Project Swing Component



Printing a Project Swing component is almost the same as printing an AWT component, except the `MyButton` subclass does not need a `paint` method implementation. It does, however, have a `print` method that calls the `paint` method for the component. The `paint` method

implementation is not needed because Project Swing components know how to paint themselves.

Here is the complete [printbutton.java](#) source code for Project Swing.

```
class MyButton extends JButton implements Printable {

    public MyButton() {
        super("MyButton");
    }

    public int print(Graphics g,
                    PageFormat pf, int pi)
        throws PrinterException {
        if (pi >= 1) {
            return Printable.NO_SUCH_PAGE;
        }

        Graphics2D g2 = (Graphics2D) g;
        g2.translate(pf.getImageableX(),
                   pf.getImageableY());
        Font f = new Font("Monospaced", Font.PLAIN,12);
        g2.setFont (f);
        paint(g2);
        return Printable.PAGE_EXISTS;
    }
}
```

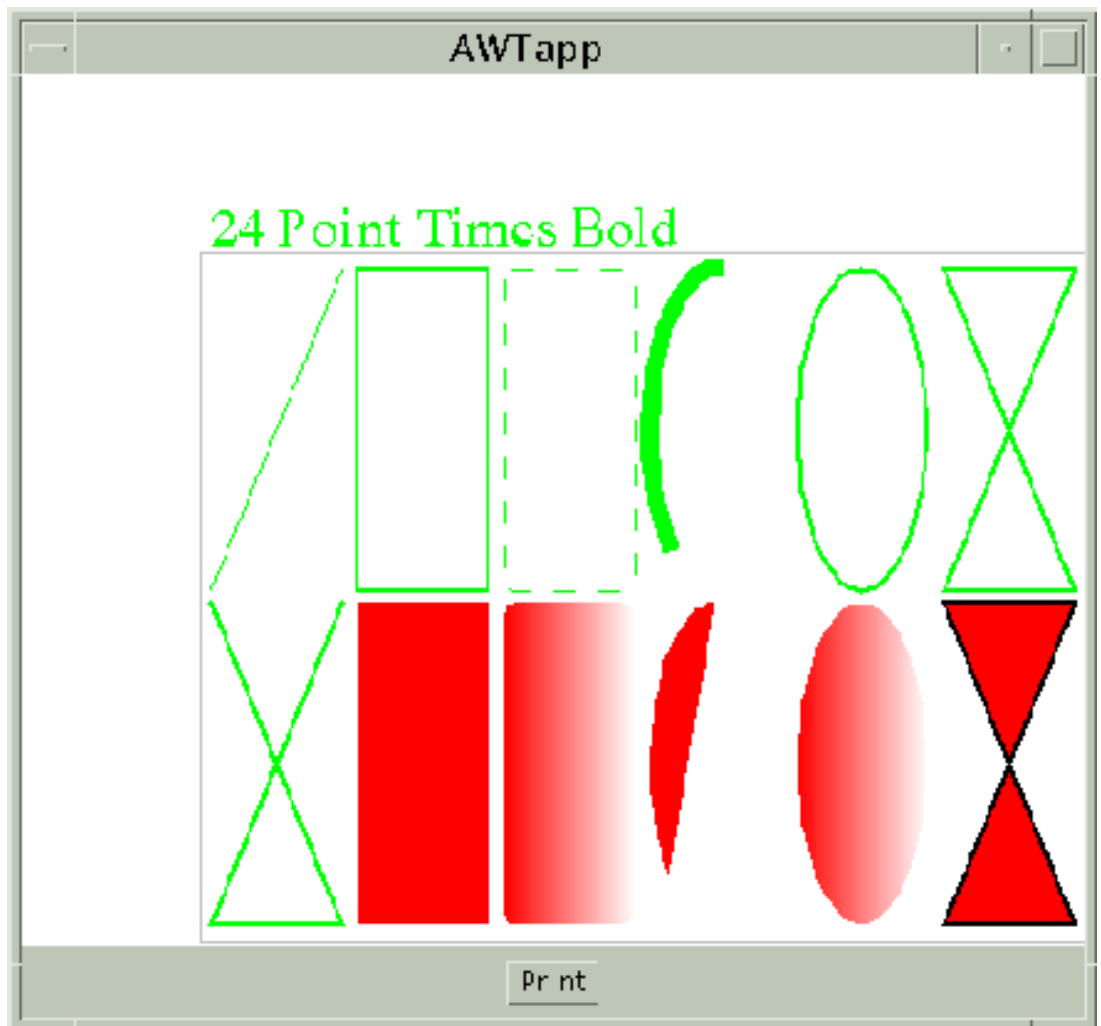
If you extend a `JPanel` and implement `Printable`, you can print a panel component and all of its contents.

```
public class printpanel extends JPanel
    implements ActionListener,
    Printable {
```

Here is the [printpanel.java](#) code that prints a `JPanel` object and the `JButton` it contains, and the [ComponentPrinterFrame.java](#) code that prints a `JFrame` object and the `JButton`, `JList`, `JCheckBox`, and `JComboBox` components it contains.

Printing Graphics in Project Swing

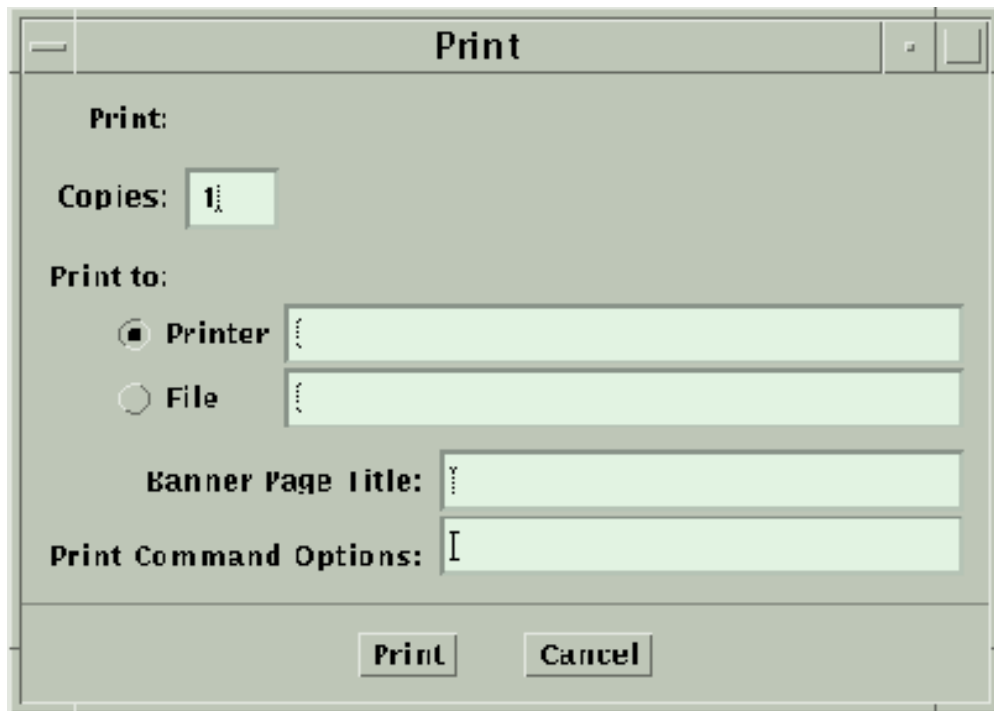
In the same way the AWT example subclassed a `Button` component and implemented the `paint` method to draw the button, you can subclass an AWT or Project Swing component and implement the `paint` method to render 2D graphics to the screen or printer. The [ShapesPrint.java](#) Project Swing application borrowed from [The Java Tutorial](#) shows how this is done. It is modified for this article to include a `TextLayout` object.



The `paintComponent` method calls the `drawShapes` method to render the 2D graphics to the screen when the application starts. When you click the `Print` button, a printer graphics context is created and passed to the `drawShapes` method for printing.

Print Dialog

It is easy to display a Print dialog so the end user can interactively change the print job properties. The `actionPerformed` method of the previous Project Swing example is modified here to do just that.



```
public void actionPerformed(ActionEvent e) {
    PrinterJob printJob = PrinterJob.getPrinterJob();
    printJob.setPrintable((MyButton) e.getSource());
    if(printJob.printDialog()){
        try { printJob.print(); }
        catch (Exception PrinterException) { }
    }
}
```

Note: In Project Swing, the

`printJob.setPageable((MyButton) e.getSource());`

statement can be written as

`printJob.setPrintable((MyButton) e.getSource());`. The difference is `setPrintable` is for applications that do not know the number of pages they are printing. If you use `setPrintable`, you need to add `if(pi >= 1){ return Printable.NO_SUCH_PAGE; }` to the beginning of the `print` method.

Page Setup Dialog

You can add a line of code that tells the `PrinterJob` object to display a Page dialog so the end user can interactively modify the page format for printing in portrait, landscape, or reverse landscape mode. The `actionPerformed` method of the previous Project Swing example is modified here to display Page and Print dialogs.

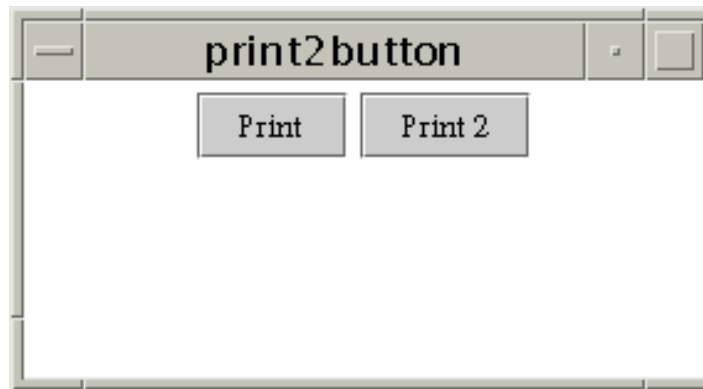
Note: Some platforms do not support a page dialog. On

those platforms, the `pageDialog` call simply returns the passed-in `PageFormat` object and no dialog appears.

```
public void actionPerformed(ActionEvent e) {
    PrinterJob printJob = PrinterJob.getPrinterJob();
    printJob.setPrintable((MyButton) e.getSource());
    PageFormat pf = printJob.pageDialog(
        printJob.defaultPage());
    if(printJob.printDialog()){
        try { printJob.print(); } catch (Exception ex) { }
    }
}
```

Printing a Collection of Pages

You can use the `Book` class to print a collection of pages that you append to the book. The pages can be in any order and have different page formats.



The [print2button.java](#) example puts the `Print` and `Print 2` buttons of type `MyButton` on a panel. It creates a book that contains the pages to print. When you click either button, the book prints one copy of the `Print` button in landscape

mode and two copies of the `Print 2` button in portrait mode, as specified in the `actionPerformed` method implementation shown here.


Note: Currently a bug restricts the Solaris platform to only print in portrait mode.

```
public void actionPerformed(ActionEvent e) {
    PrinterJob printJob = PrinterJob.getPrinterJob();

    /* Set up Book */
    PageFormat landscape = printJob.defaultPage();
    PageFormat portrait = printJob.defaultPage();
    landscape.setOrientation(PageFormat.LANDSCAPE);
    portrait.setOrientation(PageFormat.PORTRAIT);
    Book bk = new Book();
    bk.append((Printable)b, landscape);
    bk.append((Printable)b2, portrait, 2);
    printJob.setPageable(bk);
}
```

```
    try { printJob.print(); } catch (Exception ex) { }  
}
```

[\[TOP\]](#)

Printable Page 

[This page was updated: 4-Jun-2001]

[Products & APIs](#) | [Developer Connection](#) | [Docs & Training](#) | [Online Support](#)
[Community Discussion](#) | [Industry News](#) | [Solutions Marketplace](#) | [Case Studies](#)

[Glossary](#) | [Feedback](#) | [A-Z Index](#)

For more information on Java technology
and other software from Sun Microsystems, call:
(800) 786-7638
Outside the U.S. and Canada, dial your country's
[AT&T Direct Access Number](#) first.



Copyright © 1995-2001 [Sun Microsystems, Inc.](#)
All Rights Reserved. [Terms of Use](#). [Privacy Policy](#).



[Products & APIs](#)
[Developer Connection](#)
[Docs & Training](#)
[Online Support](#)
[Community Discussion](#)
[Industry News](#)
[Solutions Marketplace](#)
[Case Studies](#)

Printable Page 

■ Requires login

Early Access ■
[Downloads](#)

Bug Database ■
[Submit a Bug](#)
[View Database](#)

Newsletters
[Back Issues](#)
[Subscribe](#)

Learning Centers
[Articles](#)
[Bookshelf](#)
[Code Samples](#)
[New to Java](#)
[Question of the Week](#)
[Quizzes](#)
[Tech Tips](#)
[Tutorials](#)

Forums

[Training Index](#)

Writing Advanced Applications

Chapter 6 Continued: Advanced Printing

[\[<<BACK\]](#) [\[CONTENTS\]](#) [\[NEXT>>\]](#)

The previous section explained how to print simple components and covered techniques that can be used to print screen captures. However, if you want to print more than one component per page, or if your component is larger than one page size, you need to do some additional work inside your `print` method. This section explains what you need to do and concludes with an example of how to print the contents of a `JTable` component.

[Multiple Components Per Page](#)
[Components Larger Than One Page](#)
[Printing A `JTable` Component](#)
[Printing A Sales Report](#)

Multiple Components Per Page

There are times when printing one component on a page does not meet your printing needs. For example, you might want to include a header on each page or print a footer with the page number-- something that isn't necessarily displayed on the screen.

Unfortunately, printing multiple customized components on a page is not as easy as adding additional `paint` calls because each `paint` call overwrites the output of the previous call.

The key to printing more than one component on a page, is to use the `translate(double, double)` and `setClip` methods in the `Graphics2D` class.

The `translate` method moves an imaginary pen to the next position of the print output where the component can be painted and then printed. There are two `translate` methods in the `Graphics2D` class. To print multiple components you need the one that takes `double`

Technology Centers arguments because this `translate` method allows relative positioning. Be sure to cast any integer values to double or float. Relative positioning in this context means that previous calls to `translate` are taken into account when calculating the new translated point.

The `setClip` method is used to restrict the component to only be painted, and therefore printed, in the area specified. This lets you print multiple components on a page by moving the imaginary pen to different points on the page and then painting each component in the clip area.

Example

You can replace the `print` method in the [Abstract Window Toolkit \(AWT\)](#) and [Swing](#) `printbutton.java` examples with the following code to add the footer message *Company Confidential* to the page.

```
public int print(Graphics g, PageFormat pf, int pi)
    throws PrinterException {
    if (pi >= 1) {
        return Printable.NO_SUCH_PAGE;
    }

    Graphics2D g2 = (Graphics2D) g;
    Font f= Font.getFont("Courier");
    double height=pf.getImageableHeight();
    double width=pf.getImageableWidth();

    g2.translate(pf.getImageableX(),
        pf.getImageableY());
    g2.setColor(Color.black);
    g2.drawString("Company Confidential", (int)width/2,
        (int)height-g2.getFontMetrics().getHeight());
    g2.translate(0f,0f);
    g2.setClip(0,0,(int)width,
        (int)(height-g2.getFontMetrics().getHeight()*2));
    paint (g2);
    return Printable.PAGE_EXISTS;
}
```

In the new `print` method, the `Graphics2D` context is clipped before calling the parent `JButton` `paint` method. This prevents the `JButton` `paint` method from overwriting the bottom of the page. The `translate` method is used to point the `JButton` `paint` method to start the `paint` at offset 0,0 from the visible part of the page. The visible area was already calculated by the previous `translate` call:

```
g2.translate(pf.getImageableX(), pf.getImageableY());
```

For some components, you might also need to set the foreground color to see your results. In this example the text color was printed in black.

Useful Methods To Call In The `print` Method

The following methods are useful for calculating the number of pages required and for shrinking components to fit on a page:

PageFormat methods:

```
getImageableHeight()
```

returns the page height you can use for printing your output.

```
getImageableWidth()
```

returns the page width you can use for printing your output.

Graphics2D method:

```
scale(xratio, yratio)
```

scales the 2D graphics context by this size. A ratio of one maintains the size, less than one will shrink the graphics context.

Components Larger Than One Page

The Java™ 2 Printing API has a `Book` API that provides the concept of pages. However, the `Book` API only adds printable objects to a collection of printable objects. It does not calculate page breaks or split components over multiple pages.

When printing a simple component on a page, you only have to check for the index value being greater or equal to one and return `NO_SUCH_PAGE` when this value is reached.

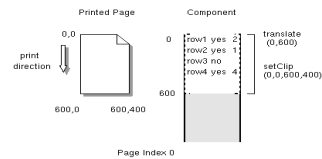
To print multiple pages, you have to calculate the number of pages needed to contain the component. You can calculate the total number of pages needed by subtracting the space taken by the component from the value returned by `getImageableHeight`. Once the total number of pages is calculated, you can run the following check inside the `print` method:

```
if (pageIndex >=TotalPages) {
    return NO_SUCH_PAGE;
}
```

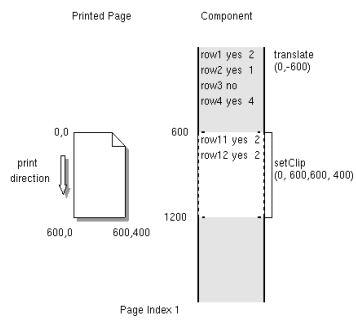
The Printing framework calls the `print` method multiple times until

`pageIndex` is less than or equal to `TotalPages`. All you need to do is create a new page from the same component on each `print` loop. This is done by treating the printed page like a sliding window over the component. The part of the component that is to be printed is selected by a `translate` call to mark the top of the page and a `setClip` call to mark the bottom of the page. The following diagram illustrates this process.

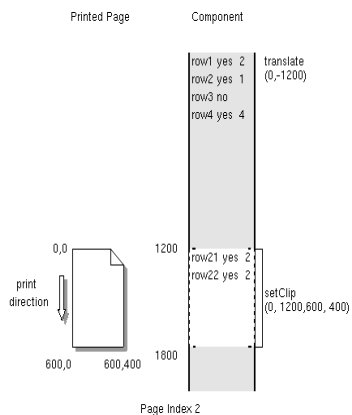
The left side of the diagram represents the page sent to the printer. The right side contains the long component being printed in the `print` method. The first page can be represented as follows:



The printed page window then slides along the component to print the second page, page index one.



This process continues until the last page from the total number of pages is reached:



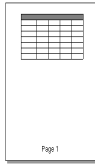
Printing A `JTable` Component

The [Report.java](#) class uses many of the advanced techniques covered in this section to print out the data and header of a `JTable` component that can span many pages. The printed output also includes a footer at the bottom with the page number.

Sales Report				
Description	open price	latest price	End Date	Quantity
Box of Biro	1.00	4.99	Mar 18, 1...	2
Blue Biro	0.10	0.14	Mar 18, 1...	1
legal pad	1.00	2.49	Mar 18, 1...	1
tape	1.00	1.49	Mar 18, 1...	1
stapler	4.00	4.49	Mar 18, 1...	1
legal pad	1.00	2.29	Mar 18, 1...	5

print me!

This diagram shows how the report looks when it prints:



```

import javax.swing.*;
import javax.swing.table.*;
import java.awt.print.*;
import java.util.*;
import java.awt.*;
import java.awt.event.*;
import java.awt.geom.*;
import java.awt.Dimension;

public class Report implements Printable{
    JFrame frame;
    JTable tableView;

    public Report() {
        frame = new JFrame("Sales Report");
        frame.addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent e) {
                System.exit(0);}});

        final String[] headers = {"Description", "open price",
            "latest price", "End Date", "Quantity"};
        final Object[][] data = {
            {"Box of Biro", "1.00", "4.99", new Date(),
                new Integer(2)},
            {"Blue Biro", "0.10", "0.14", new Date(),
                new Integer(1)},
            {"legal pad", "1.00", "2.49", new Date(),
                new Integer(1)},
            {"tape", "1.00", "1.49", new Date(),
                new Integer(1)},
            {"stapler", "4.00", "4.49", new Date(),
                new Integer(1)},
            {"legal pad", "1.00", "2.29", new Date(),
                new Integer(5)}

```

```

};

TableModel dataModel = new AbstractTableModel() {
    public int getColumnCount() {
        return headers.length; }
    public int getRowCount() { return data.length;}
    public Object getValueAt(int row, int col) {
        return data[row][col];}
    public String getColumnName(int column) {
        return headers[column];}
    public Class getColumnClass(int col) {
        return getValueAt(0,col).getClass();}
    public boolean isCellEditable(int row, int col) {
        return (col== 1);}
    public void setValueAt(Object aValue, int row,
        int column) {
        data[row][column] = aValue;
    }
};

tableView = new JTable(dataModel);
JScrollPane scrollpane = new JScrollPane(tableView);

scrollpane.setPreferredSize(new Dimension(500, 80));
frame.getContentPane().setLayout(
    new BorderLayout());
frame.getContentPane().add(
    BorderLayout.CENTER,scrollpane);
frame.pack();
JButton printButton= new JButton();

printButton.setText("print me!");

frame.getContentPane().add(
    BorderLayout.SOUTH,printButton);

// for faster printing turn double buffering off

RepaintManager.currentManager(
    frame).setDoubleBufferingEnabled(false);

printButton.addActionListener( new ActionListener(){
    public void actionPerformed(ActionEvent evt) {
        PrinterJob pj=PrinterJob.getPrinterJob();
        pj.setPrintable(Report.this);
        pj.printDialog();
        try{
            pj.print();
        } catch (Exception PrintException) {}
    }
});

frame.setVisible(true);
}

public int print(Graphics g, PageFormat pageFormat,

```

```

    int pageIndex) throws PrinterException {
        Graphics2D g2 = (Graphics2D) g;
        g2.setColor(Color.black);
        int fontHeight=g2.getFontMetrics().getHeight();
        int fontDesent=g2.getFontMetrics().getDescent();

        //leave room for page number
        double pageHeight =
            pageFormat.getImageableHeight()-fontHeight;
        double pageWidth =
            pageFormat.getImageableWidth();
        double tableWidth = (double)
            tableView.getColumnModel(
                ).getTotalColumnWidth();
        double scale = 1;
        if (tableWidth >= pageWidth) {
            scale = pageWidth / tableWidth;
        }

        double headerHeightOnPage=
            tableView.getTableHeader(
                ).getHeight()*scale;
        double tableWidthOnPage=tableWidth*scale;

        double oneRowHeight=(tableView.getRowHeight()+
            tableView.getRowMargin())*scale;
        int numRowsOnAPage=
            (int)((pageHeight-headerHeightOnPage)/
                oneRowHeight);
        double pageHeightForTable=oneRowHeight*
            numRowsOnAPage;
        int totalNumPages=
            (int)Math.ceil((
                (double)tableView.getRowCount())/
                numRowsOnAPage);
        if(pageIndex>=totalNumPages) {
            return NO_SUCH_PAGE;
        }

        g2.translate(pageFormat.getImageableX(),
            pageFormat.getImageableY());
        //bottom center
        g2.drawString("Page: "+(pageIndex+1),
            (int)pageWidth/2-35, (int)(pageHeight
                +fontHeight-fontDesent));

        g2.translate(0f,headerHeightOnPage);
        g2.translate(0f,-pageIndex*pageHeightForTable);

        //If this piece of the table is smaller
        //than the size available,
        //clip to the appropriate bounds.
        if (pageIndex + 1 == totalNumPages) {
            int lastRowPrinted =
                numRowsOnAPage * pageIndex;
            int numRowsLeft =

```

```

        tableView.getRowCount()
        - lastRowPrinted;
    g2.setClip(0,
        (int) (pageHeightForTable * pageIndex),
        (int) Math.ceil(tableWidthOnPage),
        (int) Math.ceil(oneRowHeight *
            numRowsLeft));
    }
    //else clip to the entire area available.
    else{
        g2.setClip(0,
            (int) (pageHeightForTable*pageIndex),
            (int) Math.ceil(tableWidthOnPage),
            (int) Math.ceil(pageHeightForTable));
    }

    g2.scale(scale, scale);
    tableView.paint(g2);
    g2.scale(1/scale, 1/scale);
    g2.translate(0f, pageIndex*pageHeightForTable);
    g2.translate(0f, -headerHeightOnPage);
    g2.setClip(0, 0,
        (int) Math.ceil(tableWidthOnPage),
        (int) Math.ceil(headerHeightOnPage));
    g2.scale(scale, scale);
    tableView.getTableHeader().paint(g2);
    //paint header at top

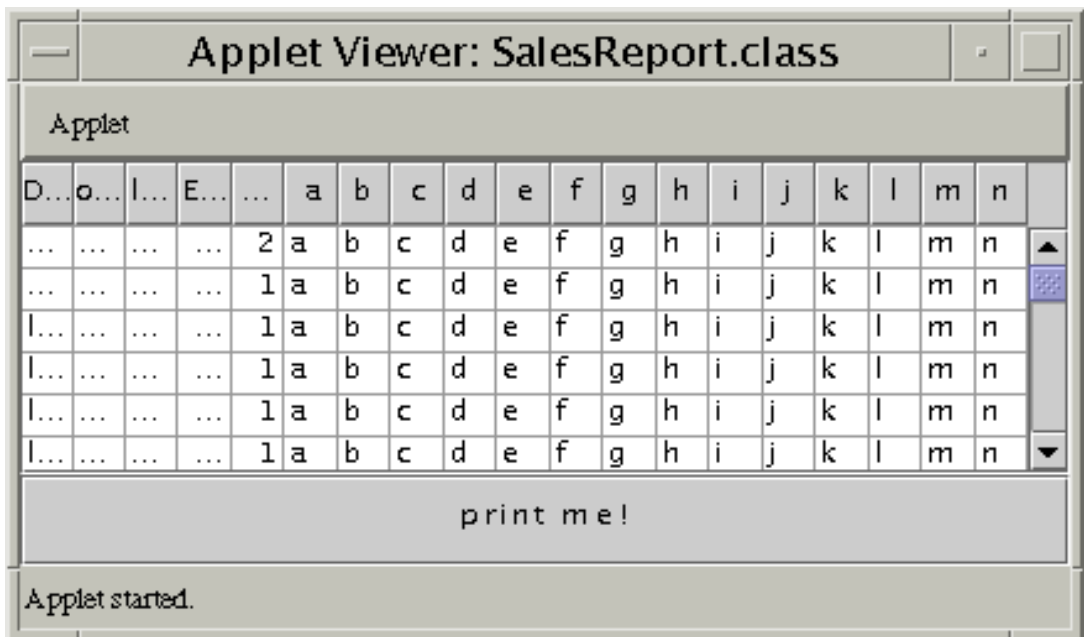
    return Printable.PAGE_EXISTS;
}

public static void main(String[] args) {
    new Report();
}
}

```

Print a Sales Report

The [SalesReport.java](#) Applet class prints a sales report with the rows split over multiple pages with numbers at the bottom of each page. Here is how the application looks when launched:



You need this policy file to launch the applet:

```
grant {
    permission java.lang.RuntimePermission
        "queuePrintJob";
};
```

To launch the applet assuming a policy file named `printpol` and an HTML file named `SalesReport.html`, you would type:

```
appletviewer -J-Djava.security.policy=
    printpol SalesReport.html
```

This diagram shows how the report prints:



[\[TOP\]](#)

[Products & APIs](#) | [Developer Connection](#) | [Docs & Training](#) | [Online Support](#)
[Community Discussion](#) | [Industry News](#) | [Solutions Marketplace](#) | [Case Studies](#)

[Glossary](#) | [Feedback](#) | [A-Z Index](#)

For more information on Java technology
and other software from Sun Microsystems, call:
(800) 786-7638
Outside the U.S. and Canada, dial your country's
[AT&T Direct Access Number](#) first.



Copyright © 1995-2001 [Sun Microsystems, Inc.](#)
All Rights Reserved. [Terms of Use](#). [Privacy Policy](#).



[Products & APIs](#)
[Developer Connection](#)
[Docs & Training](#)
[Online Support](#)
[Community Discussion](#)
[Industry News](#)
[Solutions Marketplace](#)
[Case Studies](#)

[Training Index](#)

[Printable Page](#) 

■ Requires login

[Early Access](#) ■
[Downloads](#)

[Bug Database](#) ■
[Submit a Bug](#)
[View Database](#)

[Newsletters](#)
[Back Issues](#)
[Subscribe](#)

[Learning Centers](#)
[Articles](#)
[Bookshelf](#)
[Code Samples](#)
[New to Java](#)
[Question of the Week](#)
[Quizzes](#)
[Tech Tips](#)
[Tutorials](#)

[Forums](#)

Writing Advanced Applications

Chapter 7: Debugging Applets, Applications, and Servlets

[<<BACK] [CONTENTS] [NEXT>>]

An unwritten law of programming states you will spend 10 percent of your time on the first 90 percent of a project, and the other 90 percent of your time on the remaining 10 percent. If this sounds like any of your projects, you are probably spending that last 10 percent on debugging and integration. While there are plenty of books and people to help you start a project, there are far fewer resources available to help you finish it.

The good news is this chapter focuses completely on debugging and fixing to get your project out on time. It uses real-world examples to walk you through the simple steps to debugging and fixing your programs. By the time you finish, you should be an expert at troubleshooting programs written in the Java™ language--applets, applications, and servlets--of all shapes and sizes.

[Collecting Evidence](#)
[Running Tests and Analyzing](#)
[Servlet Debugging](#)
[AWT Event Debugging](#)
[Analyzing Stack Traces](#)
[Version Issues](#)

In a Rush?

If you have a pressing problem you need an answer to right now, this table might help. It tells you where to find answers for common problems so you can go directly to the information.

Problem	Section
Program hangs or crashes	Analyzing Stack Traces

Technology Centers


[Problem in a running program](#)

[Getting Behind the Seat with jdb](#)

[Java Web Server™ problems](#)

[Servlet Debugging and Analyzing Stack Traces](#)

[\[TOP\]](#)

[Printable Page](#) 

[This page was updated: 4-Jun-2001]

[Products & APIs](#) | [Developer Connection](#) | [Docs & Training](#) | [Online Support](#)
[Community Discussion](#) | [Industry News](#) | [Solutions Marketplace](#) | [Case Studies](#)

[Glossary](#) | [Feedback](#) | [A-Z Index](#)

For more information on Java technology and other software from Sun Microsystems, call: (800) 786-7638
Outside the U.S. and Canada, dial your country's [AT&T Direct Access Number](#) first.



Copyright © 1995-2001 [Sun Microsystems, Inc.](#)
All Rights Reserved. [Terms of Use](#). [Privacy Policy](#).



[Products & APIs](#)
[Developer Connection](#)
[Docs & Training](#)
[Online Support](#)
[Community Discussion](#)
[Industry News](#)
[Solutions Marketplace](#)
[Case Studies](#)

[Training Index](#)

[Printable Page](#) 

■ Requires login

[Early Access](#) ■
[Downloads](#)

[Bug Database](#) ■
[Submit a Bug](#)
[View Database](#)

[Newsletters](#)
[Back Issues](#)
[Subscribe](#)

[Learning Centers](#)
[Articles](#)
[Bookshelf](#)
[Code Samples](#)
[New to Java](#)
[Question of the Week](#)
[Quizzes](#)
[Tech Tips](#)
[Tutorials](#)

[Forums](#)

Writing Advanced Applications

Chapter 7 Continued: Collecting Evidence

[\[<<BACK\]](#) [\[CONTENTS\]](#) [\[NEXT>>\]](#)

The first step in trying to solve any problem is to gather as much evidence and information as possible. If you can picture a crime scene, you know that everything is checked, cataloged and analyzed before any conclusions are reached. When debugging a program, you do not have weapons, hair samples, or fingerprints, but there is plenty of evidence you can gather that might contain or ultimately lead to the solution. This section explains how to gather that evidence.

[Installation and Environment](#)

[Class Path](#)

[Class Loading](#)

[Including Debug Code](#)

Installation and Environment

The Java™ platform is a fast-moving and changing technology. You might have more than one release installed on your system, and those releases might have been installed as part of another products installation. In an environment with mixed releases, a program can experience problems due to changes to the platform in a new version or release.

For example, if classes, libraries, or Windows registry entries from previous installations remain on your system after an upgrade, there is a chance the new software mix is causing your problems and needs to be investigated and ruled out. Opportunities for problems related to mixed software releases have increased with the use of different release tools to deliver the Java platform software.

The section on [Version Issues](#) at the end of this chapter provides a

Technology Centers complete list of major Java platform release and version information to help you rule out software release issues. This next section highlights the most common problems you are likely to encounter.

Class Path

In the Java 2 platform, the `CLASSPATH` environment variable is needed to specify the application's own classes only, and not the Java platform classes as was required in earlier releases. So it is possible your `CLASSPATH` environment variable is pointing at Java platform classes from earlier releases and causing problems.

To examine the `CLASSPATH`, type the following at the command line:

Windows 95/98/NT:
`echo %CLASSPATH%`

Unix Systems:
`echo $CLASSPATH`

Java classes are loaded on a first come, first served basis from the `CLASSPATH` list. If the `CLASSPATH` variable contains a reference to a `lib/classes.zip` file, which in turn points to a different Java platform installation, this can cause incompatible classes to be loaded.

Note: In the Java 2 platform, the system classes are chosen before any class on the `CLASSPATH` list to minimize the possibility of any old broken Java classes being loaded instead of a Java 2 class of the same name.

The `CLASSPATH` variable can get its settings from the command line or from configuration settings such as those specified in the User Environment on Windows NT, an `autoexec.bat` file, or a shell startup file like `.cshrc` on Unix.

You can control the classes the Java¹ Virtual Machine (VM) uses by compiling your program with a special command-line option that lets you supply the `CLASSPATH` you want. The Java 2 platform option and parameter is `-Xbootclasspath classpath`, and earlier releases use `-classpath classpath` and `-sysclasspath classpath`. Regardless of which release you are running, the `classpath` parameter specifies the system and user classpath, and zip or Java ARchive (JAR) files to be used in the compilation.

To compile and run the `Myapp.java` program with a system `CLASSPATH` supplied on the command line, use the following instructions:

Windows 95/98/NT:

In this example, the Java platform is installed in the `C:\java` directory. Type everything on one line:

```
javac -J-Xbootclasspath:c\java\lib\tools.jar;c:
\java\jre\lib\rt.jar;c:\java\jre\lib\i18n.jar;.
  Myapp.java
```

You do not need the `-J` runtime flag to run the compiled `Myapp` program, just type the following on one line:

```
java -Xbootclasspath:c:\java\jre\lib\rt.jar;c:
\java\jre\lib\i18n.jar;. Myapp
```

Unix Systems:

In this example, the Java platform is installed in the `/usr/local/java` directory. Type everything on one line:

```
javac -J-Xbootclasspath:/usr/local/java/lib/tools.jar:
/usr/local/java/jre/lib/rt.jar:
/usr/local/java/jre/lib/i18n.jar:. Myapp.java
```

You do not need the `-J` runtime flag to run the compiled `Myapp` program, just type the following on one line:

```
java -Xbootclasspath:/usr/local/java/jre/lib/rt.jar:
/usr/local/java/jre/lib/i18n.jar:. Myapp
```

Class Loading

Another way to analyze `CLASSPATH` problems is to locate where your application is loading its classes. The `-verbose` option to the `java` command shows which `.zip` or `.jar` file a class comes from when it is loaded. This way, you will be able to tell if it came from the Java platform zip file or from some other application's JAR file.

For example, an application might be using the `Password` class you wrote for it or it might be loading a `Password` class from an installed integrated development environment (IDE) tool.

You should see each jar and zip file named as in the example below:

```

$ java -verbose SalesReport
[Opened /usr/local/java/jdk1.2/solaris/jre/lib/rt.jar
  in 498 ms]
[Opened /usr/local/java/jdk1.2/solaris/jre/lib/i18n.jar
  in 60 ms]
[Loaded java.lang.NoClassDefFoundError from
  /usr/local/java/jdk1.2/solaris/jre/lib/rt.jar]
[Loaded java.lang.Class from
  /usr/local/java/jdk1.2/solaris/jre/lib/rt.jar]
[Loaded java.lang.Object from
  /usr/local/java/jdk1.2/solaris/jre/lib/rt.jar]

```

Including Debug Code

A common way to add diagnostic code to an application is to use `System.out.println` statements at strategic locations in the application. This technique is fine during development, providing you remember to remove them all when you release your product. However, there are other approaches that are just as simple, do not affect the performance of your application, and do not display messages that you do not want your customers to see. The following are two techniques that overcome the problems with simple `System.out.println` statements.

Turning Debug Information On at Runtime

The first alternative to the classic `println` debug statements is to turn on debugging information at runtime. One advantage to this is you do not need to recompile any code if problems appear at the testing stage or on a customer site.

Another advantage is that sometimes software problems can be attributed to race conditions where the same segment of code behaves unpredictably due to timing between other program interactions. If you control your debug code from the command line instead of adding `println` debug statements, you can rule out sequence problems caused by race conditions coming from the `println` code. This technique also saves you adding and removing `println` debug statements and having to recompile your code.

This technique requires you to use a system property as a debug flag and include application code to test that system property value. To turn on debug information from the command line at runtime, start the application and set the debug system property to `true` as follows:

```
java -Ddebug=true TestRuntime
```

The source code for the `TestRuntime` class needs to examine this property and set the debug boolean flag as follows:

```
public class TestRuntime {
    boolean debugmode; //global flag that we test

    public TestRuntime () {

        String dprop=System.getProperty("debug");

        if ((dprop != null) && (dprop.equals("yes"))){
            debugmode=true;
        }

        if (debugmode) {
            System.err.println("debug mode!");
        }
    }
}
```

Creating Debug and Production Releases at Compile Time

As mentioned earlier, one problem with adding `System.out.println` debug statements to your code is finding and removing them before you release the product. Apart from adding unnecessary code, `println` debug statements can contain information you do not want your customers to see.

One way to remove `System.out.println` debug statements from your code is to use the following compiler optimization to remove pre-determined branches from your code at compile time and achieve something similar to a debug pre-processor.

This example uses a static `dmode` boolean flag that when set to `false` results in the debug code and the debug test statement being removed. When the `dmode` value is set to `true`, the code is included in the compiled class file and is available to the application for debugging purposes.

```
class Debug {

    //set dmode to false to compile out debug code
    public static final boolean dmode=true;
}

public class TestCompiletime {
```

```

    if (Debug.dmode) { // These
        System.err.println("Debug message"); // are
    } // removed
}

```

Using Diagnostic Methods

You can use diagnostic methods to request debug information from the Java VM. The following two methods from the `Runtime` class trace the method calls and Java VM byte codes your application uses. As both these methods produce a lot of output, it is best to trace very small amounts of code, even as little as one line at a time.

To enable trace calls so you will see the output, you have to start the Java VM with the `java_g` or `java -Xdebug` interpreter commands.

To list every method as it is invoked at runtime, add the following line before the code you wish to start tracing and add a matching `traceMethodCalls` line with the argument set to false to turn the tracing off. The tracing information is displayed on the standard output.

```

// set boolean argument to false to disable
Runtime.getRuntime().traceMethodCalls(true);
callMyCode();
Runtime.getRuntime().traceMethodCalls(false);

```

To see each line as bytecodes as they are executed, add the following line to your application code:

```

// set boolean argument to false to disable
Runtime.getRuntime().traceInstructions(true);
callMyCode();
Runtime.getRuntime().traceInstructions(false);

```

You can also add the following line to your application to dump your own stack trace using the `dumpStack` method from the `Thread` class. The output from a stack trace is explained in [Analyzing Stack Traces](#), but for now you can think of a stack trace as a snapshot of the current threads running in the Java VM.

```

Thread.currentThread().dumpStack();

```

Adding Debug Information

Local variable information is not included in the core Java platform system classes. So, if you use a debug tool to list local variables

for system classes where you place `stop` commands, you will get the following output, even when you compile with the `-g` flag as suggested by the output. This output is from a `jdb` session:

```
main[1] locals
No local variables: try compiling with -g
```

To get access to the local variable information, you have to obtain the source (`src.zip` or `src.jar`) and recompile it with a debug flag. You can get the source for most `java.*` classes with the binary downloads from java.sun.com.

Once you download the `src.zip` or `src.jar` file, extract only the files you need. For example, to extract the `String` class, type the following at the command line:

```
unzip /tmp/src.zip src/java/lang/String.java
```

or

```
jar -xf /tmp/src.jar src/java/lang/String.java
```

Recompile the extracted class or classes with the `-g` option. You could also add your own additional diagnostics to the source file at this point.

```
javac -g src/java/lang/String.java
```

The Java 2 `javac` compiler gives you more options than just the original `-g` option for debug code, and you can reduce the size of your classes by using `-g:none`, which gives you on average about a 10 percent reduction in size.

To run the application with the newly compiled debug class or classes, you need to use the `bootclasspath` option so these new classes are picked up first.

Type the following on one line with a space before `myapp`.

Win95/NT Java 2 Platform:

This example assumes the Java platform is installed in `c:\java`, and the source files are in `c:\java\src`:

```
jdb -Xbootclasspath:c:\java\src;c:\java\jre\lib\rt.jar;c:\
\java\jre\i18n.jar;. myapp
```


Unix Systems:


This example assumes the Java platform is installed in `c:\java`, and the source files are in `c:\java\src`.

```
jdb -Xbootclasspath:/usr/java/src;  
/usr/java/jre/lib/rt.jar;  
/usr/java/jre/i18n.jar;. myapp
```

The next time you run the `locals` command you will see the internal fields of the class you wish to analyze.

[\[TOP\]](#)

¹ As used on this web site, the terms "Java virtual machine" or "JVM" mean a virtual machine for the Java platform.

Printable Page 

[This page was updated: 4-Jun-2001]

[Products & APIs](#) | [Developer Connection](#) | [Docs & Training](#) | [Online Support](#)
[Community Discussion](#) | [Industry News](#) | [Solutions Marketplace](#) | [Case Studies](#)

[Glossary](#) | [Feedback](#) | [A-Z Index](#)

For more information on Java technology and other software from Sun Microsystems, call: (800) 786-7638
Outside the U.S. and Canada, dial your country's [AT&T Direct Access Number](#) first.



Copyright © 1995-2001 [Sun Microsystems, Inc.](#)
All Rights Reserved. [Terms of Use](#). [Privacy Policy](#).



[Products & APIs](#)
[Developer Connection](#)
[Docs & Training](#)
[Online Support](#)
[Community Discussion](#)
[Industry News](#)
[Solutions Marketplace](#)
[Case Studies](#)

[Training Index](#)

Writing Advanced Applications

Chapter 7 Continued: Running Tests and Analyzing

[\[<<BACK\]](#)
[\[CONTENTS\]](#)
[\[NEXT>>\]](#)

If you are still having problems even after you have ruled out installation and environment problems and included debugging code, it is time to use tools to test and analyze your program.

[Printable Page](#) 

■ Requires login

[Early Access](#) ■
[Downloads](#)

[Bug Database](#) ■
[Submit a Bug](#)
[View Database](#)

[Newsletters](#)
[Back Issues](#)
[Subscribe](#)

[Learning Centers](#)
[Articles](#)
[Bookshelf](#)
[Code Samples](#)
[New to Java](#)
[Question of the Week](#)
[Quizzes](#)
[Tech Tips](#)
[Tutorials](#)

[Forums](#)

[Getting Behind the Seat with jdb](#)

[Simple jdb Test Drive](#)

[Remote Debugging](#)

[Using Auto-Pilot](#)

[Creating a Session Log](#)

Getting Behind the Seat with jdb

Although there are some very good integrated development environment (IDE) tools on the market, the Java™ debugger tool, jdb and its successors have an important role to play in testing and debugging programs. Some advantages of jdb over IDE tools are it is free, it is platform independent (some IDE tools are not), and it runs as a separate process to the program it is debugging. The benefit to jdb running as a separate process is you can attach a debug session to a running program.

The downsides to using jdb are it has only a command-line interface, and it relies on the same code you are trying to debug. This means if there is a bug in the Java VM, jdb could break attempting to diagnose that same bug!

The new JBUG architecture was created to solve these problems in jdb. JBUG, amongst other things, provides a debugger helper API in the Java VM called the Java VM Debug Interface (JVMDI). This helper communicates with the debugging front end using the Java Debug Wire Protocol (JDWP). The debugging front end uses the remote Java Debug Interface (JDI) to send and receive commands

Technology Centers over the Java Debug Wire Protocol. JBug is available for Java 2 platforms, and has a `jdb` style front end that you will learn more about later.

Simple `jdb` Test Drive

Back to the classic `jdb` tool. Here are some simple steps to analyze a program using `jdb`. This first example debugs a program from application startup. The [Remote Debugging](#) example shows how to connect to a running program.

Start the Session

To begin the debug session, compile the [SimpleJdbTest.java](#) program with full debugging information using `javac` and the `-g` debug flag as shown. In this example, the `SimpleJdbTest.java` program is an application, but it could just as well be an applet. The procedures for debugging applications with `jdb` are the same for debugging applets once the debug session has started.

```
javac -g SimpleJdbTest.java
```

Next, start the `jdb` tool with the program class name as a parameter:

```
jdb SimpleJdbTest
Initializing jdb...
0xad: class(SimpleJdbTest)
```

To debug an applet in `appletviewer` use the `-debug` parameter as in this example:

```
$ appletviewer -debug MyApplet.html
Initializing jdb...
0xee2f9808: class(sun.applet.AppletViewer)
>
```

Setting a Breakpoint and Listing Methods

At this point, the `SimpleJdbTest` class has only been loaded; the class constructor has not been called. To make `jdb` stop when the program is first instantiated, put a stop, or breakpoint, at the constructor using the `stop in` command. When the breakpoints has been set, instruct `jdb` to run your program using the `run` command as follows:

```
stop in SimpleJdbTest.<init>
Breakpoint set in SimpleJdbTest.<init>
run
```

```

run SimpleJdbTest
running ...
main[1]
Breakpoint hit: SimpleJdbTest.<init>
                (SimpleJdbTest:10)

```

The `jdb` tool stops at the first line in the constructor. To list the methods that were called to get to this breakpoint, enter the `where` command:

```

main[1] where
[1] SimpleJdbTest.<init> (SimpleJdbTest:10)
[2] SimpleJdbTest.main (SimpleJdbTest:29)

```

The numbered method in the list is the last stack frame that the Java VM has reached. In this case the last stack frame is the `SimpleJdbTest` constructor that was called from `SimpleJdbTest` `main`.

Whenever a new method is called, it is placed on this stack list. The Hotspot technology achieves some of its speed gains by eliminating a new stack frame when a new method is called. To gain a general appreciation of where the code has stopped, enter the `list` command.

```

main[1] list
6           Panel p;
7           Button b;
8           int counter=0;
9
10          SimpleJdbTest() {
11              setSize(100,200);
12              setup();
13          }
14          void setup () {

```

Locating the Source

If the source to the class file stopped in is not available on the current path, you can tell `jdb` to find the source with the `use` command by giving it the source directory as a parameter. In the following example the source is in a subdirectory or folder called `book`.

```

main[1] list
Unable to find SimpleJdbTest.java
main[1] use book
main[1] list
6           Panel p;
7           Button b[];
8           int counter=0;
9
10          => SimpleJdbTest() {

```

Looking at a Method

To see what happens in the `setup` method for `SimpleJdbText`, use the `step` command to step through the 4 lines to get to it.

```
main[1] step
main[1]
Breakpoint hit: java.awt.Frame.<init> (Frame:222)
```

But wait a minute! This is now the `Frame` class constructor! If you keep stepping you follow the `Frame` Constructor and not the `SimpleJdbText` class. Because `SimpleJdbText` extends the `Frame` class, the parent constructor, which in this case is `Frame`, is called on your behalf.

The `step up` Command

You could continue stepping and eventually you will return to the `SimpleJdbText` constructor, but to return immediately, you can use the `step up` command to go back to the `SimpleJdbText` constructor.

```
main[1] step up
main[1]
Breakpoint hit: SimpleJdbTest.<init>
                (SimpleJdbTest:8)
```

The `next` Command

You can also use the `next` command to get to the `setup` method. In this next example, the `jdb` tool has approximated that the source line is outside the constructor when it processed the last `step up` command. To return to the constructor, use another `step` command, and to get to the `setup` method, use a `next` command. To debug the `setup` method, you can step through the `setup` method.

```
main[1] step
Breakpoint hit: SimpleJdbTest.<init>
                (SimpleJdbTest:11)

main[1] list
7           Button b[]=new Button[2];
8           int counter=0;
9
10          SimpleJdbTest() {
11              setSize(100,200);<
12              setup();
13          }
14          void setup (){
15              p=new Panel();
16          }
main[1] next
Breakpoint hit: SimpleJdbTest.<init>
                (SimpleJdbTest:12)

main[1] step
```

```
Breakpoint hit: SimpleJdbTest.setup (SimpleJdbTest:15)
```

The `stop in` Command

Another way to get to the `setup` method is to use the `stop in SimpleJdbTest.setup` command. You can list the source again to check where you are:

```
main[1] list
11             setSize(100,200);
12             setup();
13         }
14     void setup (){
15     =>         p=new Panel();
16             b[0]= new Button("press");
17             p.add(b[0]);
18             add(p);
19
```

The `print` Command

The first thing the `setup` method does is create a `Panel p`. If you try to display the value of `p` with the `print p` command, you will find that the value is `null`.

```
main[1] print p
p = null
```

This occurred because the line has not been executed and so field `p` has not been assigned a value. You need to step over that assignment operation with the `next` command and then use the `print p` command again.

```
main[1] next
```

```
Breakpoint hit: SimpleJdbTest.setup (SimpleJdbTest:16)
main[1] print p
p = java.awt.Panel[panel0,0,0,0x0,invalid,
    layout=java.awt.FlowLayout]
```

Setting Breakpoints on Overloaded Methods

Although stepping through small classes is fast, as a general rule on larger applications, it is often a lot faster to set breakpoints. This is partly because `jdb` has a very simple command set and no shortcuts, so each command has to be pasted or typed in full.

To set a breakpoint in the `Button` class, use `stop in java.awt.Button.<init>`

```
main[1] stop in java.awt.Button.<init>
java.awt.Button.<init> is overloaded,
    use one of the following:
void <init>
void <init>(java.lang.String)
```

The message explains why `jdb` cannot stop in this method without more information, but the message is slightly misleading as you do not need to specify the return type for overloaded methods, you just need to be explicit about exactly which one of the overloaded methods you want to stop in. To stop in the `Button` constructor that creates this `Button`, use `stop in java.awt.Button.<init>(java.lang.String)`.

The `cont` Command

To continue the `jdb` session, use the `cont` command. The next time the program creates a `Button` with a `String` as the constructor, `jdb` stops so you can examine the output.

```
main[1] cont
main[1]
Breakpoint hit: java.awt.Button.<init>
                (Button: 130)
```

If the `Button` class had not been recompiled with debug information as described earlier, you would not see the internal fields from the `print` command.

Clearing Breakpoints

To clear this breakpoint and not stop every time a `Button` is created use the `clear` command. This example uses the `clear` command with no arguments to display the list of current breakpoints, and the `clear` command with the `java.awt.Button:130.` argument to clear the `java.awt.Button:130.` breakpoint.

```
main[1] clear
Current breakpoints set:
SimpleJdbTest:10
java.awt.Button:130
main[1] clear java.awt.Button:130
Breakpoint cleared at java.awt.Button: 130
```

Displaying Object Details

To display details about an object, use the `print` command to call the object's `toString` method, or use the `dump` command to display

the object's fields and values.

This example puts a breakpoint at line 17 and uses the `print` and `dump` commands to print and dump the first `Button` object in the array of `Button` objects. The `dump` command output has been abbreviated.

```
main[1] stop at SimpleJdbTest:17
Breakpoint set at SimpleJdbTest:17
main[1] cont
main[1]
Breakpoint hit: SimpleJdbTest.setup (SimpleJdbTest:17)

main[1] print b[0]
b[0] = java.awt.Button[button1,0,0,0x0,invalid,
                                label=press]

main[1] dump b[0]
b[0] = (java.awt.Button)0x163 {
private int componentSerializedDataVersion = 2
boolean isPacked = false
private java.beans.PropertyChangeSupport
        changeSupport = null
long eventMask = 4096
transient java.awt.event.InputMethodListener
        inputMethodListener = null
....
java.lang.String actionCommand = null
java.lang.String label = press
}
```

Ending the Session

That finishes the simple `jdb` examples. To terminate the `jdb` session, use the `quit` command:

```
0xee2f9820: class(SimpleJdbTest)
> quit
```

Remote Debugging

The `jdb` tool is an external process debugger, which means it debugs the program by sending messages to and from a helper inside the Java VM. This makes it is easy to debug a running program, and helps you debug a program that interacts with the end user. A remote debug session from the command-line does not interfere with the normal operation of the application.

Starting the Session

Before the Java 2 release, the only thing required to enable remoted debugging was to start the program with the `-debug` flag as the first argument, and if the application uses native libraries, make the library name end in `_g`. For example, you would need to

copy `nativelib.dll` to `nativelib_g.dll` to debug with that library.

In Java 2, things are a little more complicated. You need to tell the Java VM where the `tools.jar` file is by using the `CLASSPATH` variable. The `tools.jar` file is normally found in the `lib` directory of the Java platform installation.

You also need to disable the Just In Time (JIT) compiler if one exists. The JIT compiler is disabled by setting the `java.compiler` property to `NONE` or to an empty string. Finally, as the `-classpath` option overrides any previously set user classpath, you also need to add the `CLASSPATH` needed by your application.

Putting all of this together, here is the command line needed to start a program in remote debug mode. Put this all on one line and include all the classes you need on the command line.

Windows:

```
$ java -debug -classpath C:\java\lib\tools.jar;.
-Djava.compiler=NONE SimpleJdbTest
Agent password=4gk5hm
```

Unix:

```
$ java -debug -classpath /usr/java/lib/tools.jar:.
-Djava.compiler=NONE SimpleJdbTest
Agent password=5ufhic
```

The output is the agent password (in this case, `4gk5hm`) if the program was successfully started. The agent password is supplied when starting `jdb` so `jdb` can find the corresponding application started in debug mode on that machine.

To start `jdb` in remote debug mode, supply a host name, which can be either the machine where the remote program was started or `localhost` if you are debugging on the same machine as the remote program, and the agent password.

```
jdb -host localhost -password 4gk5hm
```

Listing Threads

Once inside the `jdb` session, you can list the currently active threads with the `threads` command, and use the `thread <threadnumber>` command, for example, `thread 7` to select the thread to analyze. Once the thread is selected, use the `where` command to see which methods have been called for this thread.

```
$ jdb -host arsenal -password 5ufhic
```

```

Initializing jdb...
> threads
Group system:
1. (java.lang.Thread)0x9      Signal dispatcher
   cond. waiting
2. (java.lang.ref.Reference  0xb Reference Handler
   $ReferenceHandler)        cond. waiting
3. (java.lang.ref.           Finalizer
   Finalizer                  cond. waiting
   $FinalizerThread)0xd
4. (java.lang.Thread)0xe     Debugger agent
   running
5. (sun.tools.agent.         Breakpoint handler
   Handler)0x10              cond. waiting
6. (sun.tools.agent.         Step handler
   StepHandler)0x12         cond. waiting
Group main:
7. (java.awt.                AWT-EventQueue-0
   EventDispatchThread)     cond. waiting
   0x19
8. (sun.awt.                 PostEventQueue-0
   PostEventQueue)0x1b      cond. waiting
9. (java.lang.Thread)0x1c    AWT-Motif
   running
10. (java.lang.Thread)0x1d   TimerQueue
   cond. waiting
11. (sun.awt.                Screen Updater
   ScreenUpdater)0x1f       cond. waiting
12. (java.lang.Thread)0x20   Thread-0
   cond. waiting
> thread 7
AWT-EventQueue-0[1] where
[1] java.lang.Object.wait (native method)
[2] java.lang.Object.wait (Object: 424)
[3] java.awt.EventQueue.getNextEvent
   (EventQueue: 179)
[4] java.awt.EventDispatchThread.run
   (EventDispatchThread: 67)

```

Listing Source

To list the source, the thread needs to be suspended using the `suspend` command. To let this thread continue use the `resume` command. The example uses `resume 7`.

```

AWT-EventQueue-0[1] suspend 7
AWT-EventQueue-0[1] list
Current method is native
AWT-EventQueue-0[1] where
[1] java.lang.Object.wait (native method)
[2] java.lang.Object.wait (Object: 424)

```

```
[3] java.awt.EventQueue.getNextEvent
      (EventQueue:179)
[4] java.awt.EventDispatchThread.run
      (EventDispatchThread:67)
AWT-EventQueue-0[1] resume 7
```

Ending the Session

When you finish debugging this program remotely, clear any remaining breakpoints before quitting the debug session. To get a list of remaining breakpoints use the `clear` command, and to remove them enter `clear class:linenumber` as follows:

```
main[1] clear
Current breakpoints set:
SimpleJdbTest:10

main[1] clear SimpleJdbTest:10
main[1] quit
```

Using Auto-Pilot

One little known trick with `jdb` is the `jdb` startup file. `jdb` automatically looks for a file called `jdb.ini` in the `user.home` directory. If you have multiple projects, it is a good idea to set a different `user.home` property for each project when you start `jdb`. To start `jdb` with a `jdb.ini` file in the current directory, type the following:

```
jdb -J-Duser.home=.
```

The `jdb.ini` file lets you set up `jdb` configuration commands, such as `use`, without having to enter the details each time `jdb` runs. The following example `jdb.ini` file starts a `jdb` session for the `FacTest` class. It includes the Java platform sources on the source path list and passes the parameter 6 to the program. It then runs and stops at line 13, displays the free memory, and waits for further input.

```
load FacTest
stop at FacTest:13
use /home/calvin/java:/home/calvin/jdk/src/
run FacTest 6
memory
```

Here is the output from the `jdb.ini` file execution:

```
$ jdb -J-Duser.home=/home/calvin/java
Initializing jdb...
0xad:class(FacTest)
Breakpoint set at FacTest:13
```

```

running ...
Free: 662384, total: 1048568
main[1]
Breakpoint hit: FacTest.compute (FacTest:13)
main[1]

```

You might wonder if `jdb.ini` files can be used to control an entire `jdb` session. Unfortunately, commands in a `jdb.ini` startup file are executed synchronously, and `jdb` does not wait until a breakpoint is reached before executing the next command. This makes printing variables awkward. You can add artificial delays with repeated `help` commands, but there is still no guarantee the thread will be suspended when you need it to be.

Creating a Session Log

You can use a little-known `jdb` feature to obtain a record of your debug session. The output is similar to what you see when you run `jdb -dbgtrace`.

To enable `jdb` logging, create a file called `.agentLog` in the directory where you are running `jdb` or `java -debug`. In the `.agentLog` file, put the file name that you want the session information to be written to on the first line. For example, an `.agentLog` file would have these contents:

```
jdblog
```

When you next run `jdb` or `java -debug`, you will see `jdb` session information as shown below. You can use this information to retrieve the breakpoint hits and the commands entered if you need to reproduce this debug session.

```

---- debug agent message log ----
[debug agent: adding Debugger agent to
system thread list]
[debug agent: adding Breakpoint handler
to system thread list]
[debug agent: adding Step handler to
system thread list]
[debug agent: adding Finalizer to
system thread list]
[debug agent: adding Reference Handler to
system thread list]
[debug agent: adding Signal dispatcher to
system thread list]
[debug agent: Awaiting new step request]
[debug agent: cmd socket:
Socket[addr=localhost/127.0.0.1,
port=38986,localport=3 8985]]

```

```
[debug agent: connection accepted]
[debug agent: dumpClasses()]
[debug agent: no such class: HelloWorldApp.main]
[debug agent: Adding breakpoint bkpt:main(0)]
[debug agent: no last suspended to resume]
[debug agent: Getting threads for HelloWorldApp.main]
```

[\[TOP\]](#)

Printable Page 

[This page was updated: 4-Jun-2001]

[Products & APIs](#) | [Developer Connection](#) | [Docs & Training](#) | [Online Support](#)
[Community Discussion](#) | [Industry News](#) | [Solutions Marketplace](#) | [Case Studies](#)

[Glossary](#) | [Feedback](#) | [A-Z Index](#)

For more information on Java technology
and other software from Sun Microsystems, call:
(800) 786-7638
Outside the U.S. and Canada, dial your country's
[AT&T Direct Access Number](#) first.



Copyright © 1995-2001 [Sun Microsystems, Inc.](#)
All Rights Reserved. [Terms of Use](#). [Privacy Policy](#).



[Products & APIs](#)
[Developer Connection](#)
[Docs & Training](#)
[Online Support](#)
[Community Discussion](#)
[Industry News](#)
[Solutions Marketplace](#)
[Case Studies](#)

[Training Index](#)

[Printable Page](#) 

■ Requires login

[Early Access](#) ■

[Downloads](#)

[Bug Database](#) ■

[Submit a Bug](#)

[View Database](#)

[Newsletters](#)

[Back Issues](#)

[Subscribe](#)

[Learning Centers](#)

[Articles](#)

[Bookshelf](#)

[Code Samples](#)

[New to Java](#)

[Question of the Week](#)

[Quizzes](#)

[Tech Tips](#)

[Tutorials](#)

[Forums](#)

Writing Advanced Applications

Chapter 7 Continued: Servlet Debugging

[\[<<BACK\]](#) [\[CONTENTS\]](#) [\[NEXT>>\]](#)

You can debug servlets with the same `jdb` commands you use to debug an applet or an application. The Java™ Servlet Development Kit (JSDK) provides a standalone program called `servletrunner` that lets you run a servlet without a web browser. On most systems, this program simply runs the `java sun.servlet.http.HttpServer` command. You can, therefore, start a `jdb` session with the `HttpServer` class.

A key point to remember when debugging servlets is that Java Web server and `servletrunner` achieve servlet loading and unloading by not including the `servlets` directory on the `CLASSPATH`. This means the servlets are loaded using a custom classloader and not the default system classloader.

[Running servletrunner in Debug Mode](#)

[Running Java Web Server™ in Debug Mode](#)

Running servletrunner in Debug Mode

In this example, the `servlets examples` directory is included on the `CLASSPATH`. You can configure the `CLASSPATH` for debug mode as follows:

Unix

```
$ export CLASSPATH=./lib/jsdk.jar:./examples:$CLASSPATH
```

Windows

```
$ set CLASSPATH=lib\jsdk.jar;examples;%classpath%
```

To start the `servletrunner` program you can either run the supplied startup script called `servletrunner` or just supply the `servletrunner` classes as a parameter to `jdb`. This example uses the parameter to

Technology Centers `servletrunner`.

```

$ jdb sun.servlet.http.HttpServer
Initializing jdb...
0xee2fa2f8:class(sun.servlet.http.HttpServer)
> stop in SnoopServlet.doGet
Breakpoint set in SnoopServlet.doGet
> run
run sun.servlet.http.HttpServer
running ...
main[1] servletrunner starting with settings:
port = 8080
backlog = 50
max handlers = 100
timeout = 5000
servlet dir = ./examples
document dir = ./examples
servlet propfile = ./examples/servlet.properties

```

To run SnoopServlet in debug mode, enter the following URL in a browser where yourmachine is the machine where you started servlet runner and 8080 is the port number displayed in the settings output.

```
http://yourmachine:8080/servlet/SnoopServlet
```

In this example jdb stops at the first line of the servlet's doGet method. The browser will wait for a response from your servlet until a timeout is reached.

```

main[1] SnoopServlet: init

Breakpoint hit: SnoopServlet.doGet (SnoopServlet:45)
Thread-105[1]

```

We can use the list command to work out where jdb has stopped in the source.

```

Thread-105[1] list
41     throws ServletException, IOException
42     {
43     PrintWriter    out;
44
45 =>   res.setContentType("text/html");
46     out = res.getWriter ();
47
48     out.println("<html>");
49     out.println("<head>
                                <title>Snoop Servlet
                                </title></head>");
Thread-105[1]

```

The servlet can continue using the cont command.

Thread-105[1] cont

Running Java Web Server in Debug Mode

The JSDK release does not contain classes available in the Java Web server and it also has its own special servlet configuration. If you cannot run your servlet from `servletrunner`, then the other option is to run the Java Web server in debug mode.

To do this add the `-debug` flag for the first parameter after the `java` program. For example in the script `bin/js` change the `JAVA` line to look like the following. In releases prior to the Java 2 platform release, you will also need to change the program pointed to by the variable `$JAVA` to `java_g` instead of `java`.

Before:

```
exec $JAVA $THREADS $JITCOMPILER $COMPILER $MS $MX \
```

After:

```
exec $JAVA -debug $THREADS $JITCOMPILER
                                $COMPILER $MS $MX \
```

Here is how to remotely connect to the Java Web Server. The agent password is generated on the standard output from the Java Web Server so it can be redirected into a file somewhere. You can find out where by checking the Java Web Server startup scripts.

```
jdb -host localhost -password <the agent password>
```

The servlets are loaded by a separate classloader if they are contained in the `servlets` directory, which is not on the `CLASSPATH` used when starting the Java Web server. Unfortunately, when debugging remotely with `jdb`, you cannot control the custom classloader and request it to load the servlet, so you have to either include the `servlets` directory on the `CLASSPATH` for debugging or load the servlet by requesting it through a web browser and then placing a breakpoint once the servlet has run.

In this next example, the `jdbc.WebServer.PasswordServlet` is included on the `CLASSPATH` when Java Web server starts. The example sets a breakpoint to stop in the `service` method of this servlet, which is the main processing method of this servlet.

The Java Web Server standard output produces this message, which lets you proceed with the remote `jdb` session:


```

Agent password=3yg23k

$ jdb -host localhost -password 3yg23k
Initializing jdb...
> stop in jdc.WebServer.PasswordServlet:service
Breakpoint set in jdc.WebServer.PasswordServlet.service
> stop
Current breakpoints set:
    jdc.WebServer.PasswordServlet:111

```

The second `stop` lists the current breakpoints in this session and shows the line number where the breakpoint is set. You can now call the servlet through your HTML page. In this example, the servlet is run as a POST operation

```

<FORM METHOD="post" action="/servlet/PasswordServlet">
<INPUT TYPE=TEXT SIZE=15 Name="user" Value="">
<INPUT TYPE=SUBMIT Name="Submit" Value="Submit">
</FORM>

```

You get control of the Java Web Server thread when the breakpoint is reached, and you can continue debugging using the same techniques as used in the [Remote Debugging](#) section.

```

Breakpoint hit: jdc.WebServer.PasswordServlet.service
(PasswordServlet:111) webpageservice Handler[1] where
[1] jdc.WebServer.PasswordServlet.service
    (PasswordServlet:111)
[2] javax.servlet.http.HttpServlet.service
    (HttpServlet:588)
[3] com.sun.server.ServletState.callService
    (ServletState:204)
[4] com.sun.server.ServletManager.callServletService
    (ServletManager:940)
[5] com.sun.server.http.InvokerServlet.service
    (InvokerServlet:101)

```

A common problem when using the Java WebServer and other servlet environments is that Exceptions are thrown but are caught and handled outside the scope of your servlet. The `catch` command allows you to trap all these exceptions.

```

webpageservice Handler[1] catch java.io.IOException
webpageservice Handler[1]
Exception: java.io.FileNotFoundException
at com.sun.server.http.FileServlet.sendResponse(
    FileServlet.java:153)
at com.sun.server.http.FileServlet.service(
    FileServlet.java:114)
at com.sun.server.webserver.FileServlet.service(
    FileServlet.java:202)

```

```
at javax.servlet.http.HttpServlet.service(  
    HttpServlet.java: 588)  
at com.sun.server.ServletManager.callServletService(  
    ServletManager.java: 936)  
at com.sun.server.webserver.HttpServiceHandler  
    .handleRequest(HttpServiceHandler.java: 416)  
at com.sun.server.webserver.HttpServiceHandler  
    .handleRequest(HttpServiceHandler.java: 246)  
at com.sun.server.HandlerThread.run(  
    HandlerThread.java: 154)
```

This simple example was generated when the file was not found, but this technique can be used for problems with posted data. Remember to use `cont` to allow the web server to proceed. To clear this trap use the `ignore` command.

```
webpageservice Handler[1] ignore java.io.IOException  
webpageservice Handler[1] catch  
webpageservice Handler[1]
```

[\[TOP\]](#)

Printable Page 

[This page was updated: 4-Jun-2001]

[Products & APIs](#) | [Developer Connection](#) | [Docs & Training](#) | [Online Support](#)
[Community Discussion](#) | [Industry News](#) | [Solutions Marketplace](#) | [Case Studies](#)

[Glossary](#) | [Feedback](#) | [A-Z Index](#)

For more information on Java technology
and other software from Sun Microsystems, call:
(800) 786-7638
Outside the U.S. and Canada, dial your country's
[AT&T Direct Access Number](#) first.



Copyright © 1995-2001 [Sun Microsystems, Inc.](#)
All Rights Reserved. [Terms of Use](#). [Privacy Policy](#).



[Products & APIs](#)
[Developer Connection](#)
[Docs & Training](#)
[Online Support](#)
[Community Discussion](#)
[Industry News](#)
[Solutions Marketplace](#)
[Case Studies](#)

Printable Page 

■ Requires login

Early Access ■
[Downloads](#)

Bug Database ■
[Submit a Bug](#)
[View Database](#)

Newsletters
[Back Issues](#)
[Subscribe](#)

Learning Centers
[Articles](#)
[Bookshelf](#)
[Code Samples](#)
[New to Java](#)
[Question of the Week](#)
[Quizzes](#)
[Tech Tips](#)
[Tutorials](#)

Forums

[Training Index](#)

Writing Advanced Applications

Chapter 7 Continued: Abstract Window Toolkit Debugging

[\[<<BACK\]](#) [\[CONTENTS\]](#) [\[NEXT>>\]](#)

Before the new Abstract Window Toolkit (AWT) Event mechanism introduced in JDK 1.1, events were received by a component such as a `TextField`, and propagated upwards to its parent components. This meant you could simply add some diagnostic code to the component's `handleEvent` or `action` method to monitor the events as they arrived.

With the introduction of JDK 1.1 and the new system event queue, events are delivered to an event queue instead of the component itself. The events are then dispatched from the System Event queue to event listeners that register to be notified when an event has been dispatched for that object.

Using AWTEventListener

You can use an `AWTEventListener` to monitor the AWT events from a system event queue. This listener takes an event mask built from an OR operation of the `AWTEvents` you want to monitor. To obtain a simple list of the `AWTEvent` events, use the `javap -public java.awt.AWTEvent` command. This example tracks the mouse and focus events.

Note:

It is advised to not use `AWTEventListener` in a shipping product as it will degrade system performance

```
//EventTest.java
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;
```

Technology Centers

```

public class EventTest extends JFrame {


    public EventTest() {
        JButton jb1=new JButton("hello");
        getContentPane().add(jb1);

        //AWTEventListener
        getToolkit().addAWTEventListener(
            new AWTEventListener() {
                public void eventDispatched(AWTEvent e) {
                    System.out.println(e+ "\n");
                }
            }, AWTEvent.MOUSE_EVENT_MASK |
            AWTEvent.FOCUS_EVENT_MASK
        );
    }

    public static void main (String args[]) {

        EventTest et=new EventTest();
        et.setSize(300,300);
        et.pack();
        et.show();
    }
}

```

[\[TOP\]](#)Printable Page 

[This page was updated: 4-Jun-2001]

[Products & APIs](#) | [Developer Connection](#) | [Docs & Training](#) | [Online Support](#)
[Community Discussion](#) | [Industry News](#) | [Solutions Marketplace](#) | [Case Studies](#)

[Glossary](#) | [Feedback](#) | [A-Z Index](#)

For more information on Java technology
and other software from Sun Microsystems, call:
(800) 786-7638
Outside the U.S. and Canada, dial your country's
[AT&T Direct Access Number](#) first.



Copyright © 1995-2001 [Sun Microsystems, Inc.](#)
All Rights Reserved. [Terms of Use](#). [Privacy Policy](#).



[Products & APIs](#)
[Developer Connection](#)
[Docs & Training](#)
[Online Support](#)
[Community Discussion](#)
[Industry News](#)
[Solutions Marketplace](#)
[Case Studies](#)

[Training Index](#)

[Printable Page](#) 

■ Requires login

[Early Access](#) ■
[Downloads](#)

[Bug Database](#) ■
[Submit a Bug](#)
[View Database](#)

[Newsletters](#)
[Back Issues](#)
[Subscribe](#)

[Learning Centers](#)
[Articles](#)
[Bookshelf](#)
[Code Samples](#)
[New to Java](#)
[Question of the Week](#)
[Quizzes](#)
[Tech Tips](#)
[Tutorials](#)

[Forums](#)

Writing Advanced Applications

Chapter 7 Continued: Analyzing Stack Traces

[<<BACK] [CONTENTS] [NEXT>>]

Stack traces have often been considered a mystery to developers. There is little or no documentation available, and when you get one or need to generate one, time is always at a premium. The next sections uncover the secrets to debugging stack traces, and by the end, you might consider a stack trace to be a helpful tool for analyzing other programs--not just broken ones!

What is a stack trace produced by the Java™ platform? It is a user friendly snapshot of the threads and monitors in a Java¹ VM. Depending on how complex your application or applet is, a stack trace can range from fifty lines to thousands of lines of diagnostics.

Regardless of the size of the stack trace, there are a few key things that anyone can find to help diagnose most software problems, whether you are an expert or very new to the Java platform.

There are three popular ways to generate a stack trace: sending a signal to the Java VM; the Java VM generates a stack trace for you; or using debugging tools or API calls.

[Sending a Signal to the Java VM](#)

[The Java VM Generates a Stack Trace](#)

[Using Debugging Tools or API Calls Which Release Generated the Stack Trace?](#)

[Which Platform Generated the Stack Trace?](#)

[Which Thread Package Was Used?](#)

[What are the Thread States](#)

[Examining Monitors](#)

[Putting the Steps Into Practice](#)

[Expert's Checklist](#)

Technology Centers Sending a signal to the Java VM

On UNIX platforms you can send a signal to a program with the `kill` command. This is the quit signal, which is handled by the Java Virtual Machine (VM).

Unix Systems:

For example, on the Solaris™ platform, you can use the `kill -QUIT process_id` command, where `process_id` is the process number of your program.

Alternately, you can enter the key sequence `<ctrl>\` in the window where the program started.

Sending this signal instructs a signal handler in the Java VM to recursively print out all the information on the threads and monitor inside the Java VM.

Windows 95/NT:

To generate a stack trace on the Windows 95 or Windows NT platforms, enter the key sequence `<ctrl><break>` in the window where the program is running.

The Java VM Generates a Stack Trace

If the Java VM experienced an internal error such as a segmentation violation or an illegal page fault, it calls its own signal handler to print out the threads and monitor information.

Using Debugging Tools or API Calls

You can generate a partial stack trace, (which in this case is only the threads information) by using the `Thread.dumpStack` method, or the `printStackTrace` method of the `Throwable` class.

You can also obtain similar information by entering `where` inside the Java debugger.

If you are successful at generating a stack trace, you should see something similar to this [stack trace](#).

```
strings core | grep JAVA_HOME
```

In the Java 2 software release, threads that called methods resulting in a call to native code are indicated in the stack trace.

Which Release Generated The Stack Trace?

In the Java 2 release the stack trace contains the Java Virtual Machine version string, the same information you see when using the `-version` parameter.

However if there is no version string, you can still take a pretty good guess at which release this stack trace came from. Obviously, if you generated the stack trace yourself this should not be much of an issue, but you may see a stack trace posted on a newsgroup or in an email article.

First identify where the Registered Monitor Dump section is in the stack trace:

If you see a `utf8 hash table lock` in the Registered Monitor Dump, this is a Java 2 platform stack trace. The final release of the Java 2 platform also contains a version string so if a version string is missing this stack trace may be from a Java 2 beta release.

If you see a `JNI pinning lock` and no `utf8 hash lock`, this is a JDK 1.1+ release.

If neither of these appears in the Registered Monitor Dump, it is probably a JDK 1.0.2 release.

Which Platform Generated the Stack Trace?

You can also find out if the stack trace came from a Windows 95, an NT, or UNIX machine by looking for any waiting threads. On a UNIX machine the waiting threads are named explicitly. On a Windows 95, or NT machine only a count of the waiting threads is displayed:

Windows 95/NT: Finalize me queue lock: <unowned>

Writer: 1

UNIX: Finalize me queue lock: <unowned>
waiting to be notified "Finalizer Thread"

Which Thread Package was Used?

Windows 95 and Windows NT Java VMs are by default native thread Java VMs. UNIX Java VMs are by default green thread Java VMs, they use a pseudo thread implementation. To make your Java VM use native threads you need to supply the `-native` parameter, for example, `java -native MyClass`.

By verifying the existence of an `Alarm` monitor in the stack trace output you can identify that this stack trace came from a green threads Java VM.

What are the Thread States?

You will see many different threads in many different states in a snapshot from a Java VM stack trace. This table describes the various keys and their meanings.

Key	Meaning
R	Running or runnable thread
S	Suspended thread
CW	Thread waiting on a condition variable
MW	Thread waiting on a monitor lock
MS	Thread suspended waiting on a monitor lock

Normally, only threads in R, S, CW or MW should appear in the stack trace. If you see a thread in state MS, report it to Sun Microsystems, through the Java Developer ConnectionSM (JDC) Bug Parade feature, because there is a good chance it is a bug. The reason being that most of the time a thread in Monitor Wait (MW) state will appear in the S state when it is suspended.

Monitors are used to manage access to code that should only be run by a single thread at a time. Monitors are covered in more detail in the next section. The other two common thread states you may see are R, runnable threads and CW, threads in a condition wait state. Runnable threads by definition are threads that could be running or are running at that instance of time. On a multi-processor machine running a true multi-processing Operating System it is possible for all the runnable threads to be running at one time. However its more likely for the other runnable threads to be waiting on the thread scheduler to have their turn to run.

Threads in a condition wait state can be thought of as waiting for an event to occur. Often a thread will appear in state CW if it is in a `Thread.sleep` or in a synchronized wait. In our earlier stack trace our main method was waiting for a thread to complete and to be notified of its completion. In the stack trace this appears as

```
"main" (TID:0xebc981e0, sys_thread_t:0x26bb0,
                    state:CW) prio=5
  at java.lang.Object.wait(Native Method)
  at java.lang.Object.wait(Object.java:424)
  at HangingProgram.main(HangingProgram.java:33)
```


The code that created this stack trace is as follows:

```
synchronized(t1) {
    try {
        t1.wait();    //line 33
    }catch (InterruptedException e){}
}
```

In the Java 2 release monitor operations, including our wait here, are handled by the Java Virtual Machine through a JNI call to `sysMonitor`. The condition wait thread is kept on a special monitor wait queue on the object it is waiting on. This explains why even though you are only waiting on an object that the code still needs to be synchronized on that object as it is infact using the monitor for that object.

Examining Monitors

This brings us to the other part of the stack trace: the monitor dump. If you consider that the threads section of a stack trace identifies the multithreaded part of your application, then the monitors section represents the parts of your application that are single threaded.

It may be easier to imagine a monitor as a car wash. In most car washes, only one car can be in the wash at a time. In your Java code only one thread at a time can have the lock to a synchronized piece of code. All the other threads queue up to enter the synchronized code just as cars queue up to enter the car wash.

A monitor can be thought of as a lock on an object, and every object has a monitor. When you generate a stack trace, monitors are either listed as being registered or not. In the majority of cases these registered monitors, or system monitors, should not be the cause of your software problems, but it helps to be able to understand and recognize them. The following table describes the common registered monitors:

Monitor	Description
utf8 hash table	Locks the hashtable of defined i18N Strings that were loaded from the class constant pool.
JNI pinning lock	Protects block copies of arrays to native method code.

JNI global reference lock	Locks the global reference table which holds values that need to be explicitly freed, and will outlive the lifetime of the native method call.
BinClass lock	Locks access to the loaded and resolved classes list. The global table list of classes
Class linking lock	Protects a classes data when loading native libraries to resolve symbolic references
System class loader lock	Ensures that only one thread is loading a system class at a time.
Code rewrite lock	Protects code when an optimization is attempted.
Heap lock	Protects the Java heap during heap memory management
Monitor cache lock	Only one thread can have access to the monitor cache at a time this lock ensures the integrity of the monitor cache
Dynamic loading lock	Protects Unix green threads JVMs from loading the shared library stub libdl.so more than once at a time.
Monitor IO lock	Protects physical I/O for example, open and read.
User signal monitor	Controls access to the signal handler if a user signal USRSIG in green threads JVMs.
Child death monitor	Controls access to the process wait information when using the runtime system calls to run locals commands in a green threads JVM.
I/O Monitor	Controls access to the threads file descriptors for poll/select events
Alarm Monitor	Controls access to a clock handler used in green threads JVMs to handle timeouts
Thread queue lock	Protects the queue of active threads

Monitor registry	Only one thread can have access to the monitor registry at a time this lock ensures the integrity of that registry
Has finalization queue lock *	Protects the list of queue lock objects that have been garbage-collected, and deemed to need finalization. They are copied to the Finalize me queue
Finalize me queue lock *	Protects a list of objects that can be finalized at leisure
Name and type hash table lock *	Protects the JVM hash tables of constants and their types
String intern lock *	Locks the hashtable of defined Strings that were loaded from the class constant pool
Class loading lock *	Ensures only one thread loads a class at a time
Java stack lock *	Protects the free stack segments list

Note: * Lock only appeared in pre-Java 2 stack traces

The monitor registry itself is protected by a monitor. This means the thread that owns the lock is the last thread to use a monitor. It is very likely this thread is also the current thread. Because only one thread can enter a synchronized block at a time, other threads queue up at the start of the synchronized code and appear as thread state `MW`. In the monitor cache dump, they are denoted as "waiting to enter" threads. In user code a monitor is called into action wherever a synchronized block or method is used.

Any code waiting on an object or event (a wait method) also has to be inside a synchronized block. However, once the wait method is called, the lock on the synchronized object is given up.

When the thread in the wait state is notified of an event to the object, it has to compete for exclusive access to that object, and it has to obtain the monitor. Even when a thread has sent a "notify event" to the waiting threads, none of the waiting threads can actually gain control of the monitor lock until the notifying thread has left its synchronized code block.

You will see "Waiting to be notified" for threads at the wait method

Putting the Steps Into Practice

Example 1

Consider a real-life problem such as Bug ID [4098756](#), for example. You can find details on this bug in JDC Bug Parade. This bug documents a problem that occurs when using a `Choice` Component on Windows 95.

When the user selects one of the choices from the `Choice` Component using the mouse, everything is fine. However, when the user tries to use an Arrow key to move up or down the list of choices, the Java application freezes.

Fortunately, this problem is reproducible and there was a Java stack trace to help track down the problem. The full stack trace is in the bug report page, but you only need to focus on the following two key threads:

```
"AWT-Windows" (TID:0xf54b70,
sys_thread_t:0x875a80,Win32ID:0x67,
state:MW) prio= 5
java.awt.Choice.select(Choice.java:293)
sun.awt.windows.WChoicePeer.handleAction(
    WChoicePeer.java:86)
```

```
"AWT-EventQueue-0" (TID:0xf54a98,sys_thread_t:0x875c20,
Win32ID:0x8f, state:R) prio= 5
java.awt.Choice.remove(Choice.java:228)
java.awt.Choice.removeAll(Choice.java:246)
```

The `AWT-EventQueue-0` thread is in a runnable state inside the `remove` method. `Remove` is synchronized, which explains why the `AWT-Windows` thread cannot enter the `select` method. The `AWT-Windows` thread is in `MW` state (monitor wait); however, if you keep taking stack traces, this situation does not change and the graphical user interface (GUI) appears to have frozen.

This indicates that the `remove` call never returned. By following the code path to the `ChoicePeer` class, you can see this is making a native `MFC` call that does not return. That is where the real problem lies and is a bug in the Java core classes. The user's code was okay.

Example 2

In this second example you will investigate a bug that on initial outset appears to be a fault in Swing but as you will discover is due to the fact that Swing is not thread safe.

Again the bug report is available to view on the JDC site, the bug number this time is [4098525](#).

Here is a cut down sample of the code used to reproduce this problem. The modal dialog is being created from within the `JPanel` `paint` method.

```
import java.awt.event.*;
import java.awt.*;
import java.util.*;
import javax.swing.*;

class MyDialog extends Dialog
    implements ActionListener {

    MyDialog(Frame parent) {
        super(parent, "My Dialog", true);
        Button okButton = new Button("OK");
        okButton.addActionListener(this);
        add(okButton);
        pack();
    }

    public void actionPerformed(ActionEvent event) {
        dispose();
    }
}

public class Tester extends JPanel {

    MyDialog myDialog;
    boolean firstTime = true;

    public Tester (JFrame frame) throws Exception {
        super();
        myDialog = new MyDialog(frame);
    }

    void showDialogs() {
        myDialog.show();
    }

    public void paint(Graphics g) {
        super.paint(g);
        if (firstTime) {
            firstTime = false;
            showDialogs();
        }
    }
}
```

```

    }

    public static void main(String args[])
        throws Exception {

        JFrame frame = new JFrame ("Test");
        Tester gui = new Tester(frame);
        frame.getContentPane().add(gui);
        frame.setSize(800, 600);
        frame.pack();
        frame.setVisible(true);
    }
}

```

When you run this program you find that it deadlocks straight away. By taking a stack trace you see the [these key threads](#).

The stack trace you have here is slightly different to the stack trace that appears in the bug report, but caused by the same effect. We are also using the Java 2 release to generate the trace and supplied the option `-Djava.compiler=NONE` when you ran the program so that you could see the source line numbers. The thread to look for is the thread in MW, monitor wait which in this case is thread `AWT-EventQueue-1`

```

"AWT-EventQueue-1" (
  TID:0xebca8c20, sys_thread_t:0x376660,
  state:MW) prio=6
  at java.awt.Component.invalidate(Component.java:1664)
  at java.awt.Container.invalidate(Container.java:507)
  t java.awt.Window.dispatchEventImpl(Window.java:696)
  at java.awt.Component.dispatchEvent(
    Component.java:2289)
  at java.awt.EventQueue.dispatchEvent(
    EventQueue.java:258)
  at java.awt.EventDispatchThread.run(
    EventDispatchThread.java:68)

```

If you look for that line in file `java/awt/Component.java` which is contained in the `src.jar` archive, you see the following:

```

public void invalidate() {
    synchronized (getTreeLock()) { //line 1664

```

This is where our application is stuck, it is waiting for the `getTreeLock` monitor lock to become free. The next task is to find out which thread has this `getTreeLock` monitor lock held.

To see who is holding this monitor lock you look at the Monitor

cache dump and in this example you can see the following:

Monitor Cache Dump:

```
java.awt.Component$AWTTreeLock@EBC9C228/EBCF2408:
  owner "AWT-EventQueue-0" ( 0x263850) 3 entries
```

Waiting to enter:

```
"AWT-EventQueue-1" (0x376660)
```

The method `getTreeLock` monitor is actually a lock on a specially created inner class object of `AWTTreeLock`. This is the code used to create that lock in file `Component.java`.

```
static final Object LOCK = new AWTTreeLock();
static class AWTTreeLock {}
```

The current owner is `AWT-EventQueue-0`. This thread called our `paint` method to create our modal `Dialog` via a call to `paintComponent.paintComponent` itself was called from an `update` call of `JFrame`.

So where was the lock set? Well there is no simple way to find out which stack frame actually held the lock but on a simple search of `javax.swing.JComponent` you see that `getTreeLock` is called inside the method `paintChildren` which you left at line 388.

```
at Tester.paint(Tester.java:39)
at javax.swing.JComponent.paintChildren(
    JComponent.java:388)
```

The rest of the puzzle is pieced together by analyzing the `MDialogPeer.show` method. The `Dialog` code creates a new `ModalThread` which is why you see an `AWT-Modal` thread in the stack trace output, this thread is used to post the `Dialog`. It is when this event is dispatched using `AWT-EventQueue-1` which used to be the `AWT Dispatch proxy` that `getTreeLock` monitor access is required and so you have a deadlock.

Unfortunately `Swing` code is not designed to be thread safe and so the workaround in this example is to not create modal dialogs inside a `Swing` `paint` methods. Since `Swing` has to do a lot of locking and calculations as to which parts of a lightweight component needs to be painted it is strongly advised to not include synchronized code or code that will result in a synchronized call such as in a modal dialog, inside `paint` method.

This completes Java stack traces theory, and you should now know what to look for the next time you see a stack trace. To save time, you should make full use of the `JDC` bug search to see if the problem you are having has already been reported by someone

else.

Expert's Checklist

To summarize, these are the steps to take the next time you come across a problem in a Java program.


Hanging, deadlocked or frozen programs: If you think your program is hanging, generate a stack trace. Examine the threads in states `MW` or `CW`. If the program is deadlocked, some of the system threads will probably show up as the current thread because there is nothing else for the Java VM to do.

Crashed or aborted programs: On UNIX look for a core file. You can analyze this file in a native debugging tool such as `gdb` or `dbx`. Look for threads that have called native methods. Because Java technology uses a safe memory model, any corruption probably occurred in the native code. Remember that the Java VM also uses native code so it might not be a bug in your application.

Busy programs: The best course of action you can take for busy programs is to generate frequent stack traces. This will narrow down the code path that is causing the errors, and you can start your investigation from there.

[\[TOP\]](#)

¹ As used on this web site, the terms "Java virtual machine" or "JVM" mean a virtual machine for the Java platform.

Printable Page 

[This page was updated: 4-Jun-2001]

[Products & APIs](#) | [Developer Connection](#) | [Docs & Training](#) | [Online Support](#)
[Community Discussion](#) | [Industry News](#) | [Solutions Marketplace](#) | [Case Studies](#)

[Glossary](#) | [Feedback](#) | [A-Z Index](#)

For more information on Java technology and other software from Sun Microsystems, call: (800) 786-7638
 Outside the U.S. and Canada, dial your country's [AT&T Direct Access Number](#) first.



Copyright © 1995-2001 [Sun Microsystems, Inc.](#)
 All Rights Reserved. [Terms of Use](#). [Privacy Policy](#).



[Products & APIs](#)
[Developer Connection](#)
[Docs & Training](#)
[Online Support](#)
[Community Discussion](#)
[Industry News](#)
[Solutions Marketplace](#)
[Case Studies](#)

[Training Index](#)

Writing Advanced Applications

Chapter 7 Continued: Version Issues

[\[<<BACK\]](#) [\[CONTENTS\]](#) [\[NEXT>>\]](#)

[Printable Page](#) 

■ Requires login

[Early Access](#) ■
[Downloads](#)

[Bug Database](#) ■
[Submit a Bug](#)
[View Database](#)

[Newsletters](#)
[Back Issues](#)
[Subscribe](#)

[Learning Centers](#)
[Articles](#)
[Bookshelf](#)
[Code Samples](#)
[New to Java](#)
[Question of the Week](#)
[Quizzes](#)
[Tech Tips](#)
[Tutorials](#)

[Forums](#)

This section provides a table that summarizes problems and solutions related to having different versions of the Java™ platform installed on your system.

Product Deployment

JDK 1.0.2 Uses `CLASSPATH` to find and load the core system classes.

On Windows 95:

```
CLASSPATH=c:\java\lib\classes.zip
```

On Unix:

```
CLASSPATH=/usr/java/lib/classes.zip:.
```

Unix Dynamic libraries, `.dll` files, shared objects, and `.so` files are located by the `PATH` variable.

Side Effects:

The Win95 `Autoexec.bat` file contains an outdated `CLASSPATH` variable set by a user or the installation of other applications.

The WinNT User Environment contains an old `CLASSPATH` variable.

The Unix `.cshrc`, `.profile`, or `.login` scripts contains wrong `CLASSPATH`.

The `JAVA_HOME` environment variable is also used by programs so check this is not set. You can clear this field in the Bourne shell (`sh`) as follows: `unset JAVA_HOME`

Diagnostics:

Technology Centers

Use the `-classpath` option to force the Java VM to use the command-line `CLASSPATH` only: `java -classpath c:\java\lib\classes.zip;. myapp`

Product Deployment

JDK 1.1 Uses relative paths to find the `classes.zip` file from the Java platform installation. The `CLASSPATH` environment variable is used to load application classes.

Side Effects:

Other Java releases found on the application path might be picked up if the new JDK `bin` directory is not explicitly set at the front of the `PATH` environment variable.

Diagnostics:

Use the `-sysclasspath` option to force the Java VM to use the `CLASSPATH` supplied on the command line only: `java -sysclasspath c:\java\lib\classes.zip;. myapp`

Product Deployment

Java 2 Platform The platform is split into a Java Runtime Environment (JRE) and Java compiler. The JRE is included as a subdirectory in the release, and the traditional `java` and `javac` programs in the `bin` directory invoke the real program in the `jre/bin` directory. The separate `jre` launcher is no longer provided, and the `java` program is solely used instead.

The Java ARchive (JAR) files containing the core Java platform system classes, `rt.jar` and `i18.jar`, are located in the `jre/lib` directory with a relative search path.

Side Effects:

If applications previously used the `classes.zip` file to load the core Java platform systems, they might still try to load an additional set of classes in error.

Diagnostics:

Use the `-xbootclasspath` option to force the Java VM to use the `CLASSPATH` supplied on the command line only: `java -Xbootclasspath:c:\java\jre\lib\rt.jar;c:\java\jre\lib\i18n.jar;. myapp`

You might need to supply this as a runtime option as follows: `javac -J-Xbootclasspath:c:\java\lib\tools.jar;c:\java\jre\lib\rt.jar;c:\java\jre\lib\i18n.jar;. myapp`

`myapp.java`

Product Deployment

Java Plug-In On Windows 95 and Windows NT uses the registry to find installed plug-in Java platform releases.

Side Effects:

Registry can become corrupted, or plug-in removed physically but not from the registry.

Diagnostics:

Display the `java.version` and `java.class.path` property in your code and display it on the Java Plug-in Console

```
System.out.println("version="+System.getProperty(
    "java.version"
));
System.out.println("class path="+System.getProperty(
    "java.class.path"
));
```

If there is a conflict, check the registry with the `regedit` command, search for the word *VM* and if it exists, delete it and reinstall the plug-in

Product Deployment

Netscape Uses `.jar` files such as `java40.jar` in `netscape` directory.

Side Effects:

Not all Netscape releases are fully JDK 1.1 compliant. You can get upgrade patches at <http://www.netscape.com>.

Diagnostics:

Start the browser on the command line with the `-classes` option.

Product Deployment

Internet Uses .cab files to contain system classes. Also uses Explorer system registry on Windows 95/NT.

Side Effects:

Use the `regedit` command to search for the word `VM`. There is a `CLASSPATH` entry to which you can add your own classes.

Diagnostics:

The registry can become corrupted. Search for `CLASSPATH` using the `regedit` program and edit the value that `CLASSPATH` points to.

[\[TOP\]](#)

Printable Page 

[This page was updated: 4-Jun-2001]

[Products & APIs](#) | [Developer Connection](#) | [Docs & Training](#) | [Online Support](#)
[Community Discussion](#) | [Industry News](#) | [Solutions Marketplace](#) | [Case Studies](#)

[Glossary](#) | [Feedback](#) | [A-Z Index](#)

For more information on Java technology
and other software from Sun Microsystems, call:
(800) 786-7638
Outside the U.S. and Canada, dial your country's
[AT&T Direct Access Number](#) first.



Copyright © 1995-2001 [Sun Microsystems, Inc.](#)
All Rights Reserved. [Terms of Use](#). [Privacy Policy](#).



[Products & APIs](#)
[Developer Connection](#)
[Docs & Training](#)
[Online Support](#)
[Community Discussion](#)
[Industry News](#)
[Solutions Marketplace](#)
[Case Studies](#)

[Training Index](#)

Writing Advanced Applications

Chapter 8: Performance Techniques

[\[<<BACK\]](#)
[\[CONTENTS\]](#)
[\[NEXT>>\]](#)

One of the biggest challenges in developing large applications for the Java™ platform is to make the application meets its performance criteria. This chapter shows you how to track down performance bottlenecks and improve application performance.

Printable Page 

■ Requires login

Early Access ■
[Downloads](#)

Bug Database ■
[Submit a Bug](#)
[View Database](#)

Newsletters
[Back Issues](#)
[Subscribe](#)

Learning Centers
[Articles](#)
[Bookshelf](#)
[Code Samples](#)
[New to Java](#)
[Question of the Week](#)
[Quizzes](#)
[Tech Tips](#)
[Tutorials](#)

Forums

[Improving Performance by Design](#)
[Connection Pooling](#)
[Performance Features and Tools](#)
[Performance Analysis](#)
[Caching Client/Server Applications](#)

In a Rush?

This table links you directly to specific performance tuning topics.

Topic	Section
Improving Performance by Design	Improving Applet Download Speed Thread Pooling
Connection Pooling	Wrapper Classes Connection Driver Connection Pool Deadlocks and Hangs Closing Connections Example Application
Performance Features and Tools	Java VM Features JIT compilers Third-Party Tools

Technology Centers

Performance Analysis	Profiling Analyze a Program
Caching Client/Server Applications	Caching One Object Caching Many Objects

[\[TOP\]](#)

Printable Page 

[This page was updated: 4-Jun-2001]

[Products & APIs](#) | [Developer Connection](#) | [Docs & Training](#) | [Online Support](#)
[Community Discussion](#) | [Industry News](#) | [Solutions Marketplace](#) | [Case Studies](#)

[Glossary](#) | [Feedback](#) | [A-Z Index](#)

For more information on Java technology and other software from Sun Microsystems, call: (800) 786-7638
Outside the U.S. and Canada, dial your country's [AT&T Direct Access Number](#) first.



Copyright © 1995-2001 [Sun Microsystems, Inc.](#)
All Rights Reserved. [Terms of Use](#). [Privacy Policy](#).



[Products & APIs](#)
[Developer Connection](#)
[Docs & Training](#)
[Online Support](#)
[Community Discussion](#)
[Industry News](#)
[Solutions Marketplace](#)
[Case Studies](#)

[Training Index](#)

[Printable Page](#) 

■ Requires login

[Early Access](#) ■
[Downloads](#)

[Bug Database](#) ■
[Submit a Bug](#)
[View Database](#)

[Newsletters](#)
[Back Issues](#)
[Subscribe](#)

[Learning Centers](#)
[Articles](#)
[Bookshelf](#)
[Code Samples](#)
[New to Java](#)
[Question of the Week](#)
[Quizzes](#)
[Tech Tips](#)
[Tutorials](#)

[Forums](#)

Writing Advanced Applications

Chapter 8 Continued: Improving Performance By Design

[<<BACK] [CONTENTS] [NEXT>>]

Bandwidth restrictions imposed on networks around the world make network-based operations potential bottlenecks that can have a significant impact on an application's performance. Many network-based applications are designed to use connection pools so they can reuse existing network connections and save on the time and overhead invested in opening and closing network connections.

Besides connection pooling, there are other features you can design into your programs to improve performance. This chapter explains how you can design an applet to download files and resources more efficiently, or design a thread-based program to use thread pooling to save on the expensive thread startup process.

[Improving Applet Download Speed](#)
[Thread Pooling](#)

Improving Applet Download Speed

Applet download performance refers to the time it takes for the browser to download all the files and resources it needs to start the applet. An important factor affecting any applet's download performance is the number of times it has to request data from the server. You can reduce the number of requests by packaging the applet images into one class file, or using Java™ ARchive (JAR) files.

Packaging Images into One Class

Normally, if an applet has six image buttons, that translates to six additional requests sent back to the web server to load those image files. Six additional requests might not seem like much on

Technology Centers an internal network, but given connections of lesser speed and reliability, those additional requests can have a significant negative impact on performance. So, your ultimate goal should be to load the applet as quickly as possible.

One way to store images in a class file is to use an ASCII encoding scheme such as [X-PixMap \(XPM\)](#). This way, rather than maintaining the images as GIF files on the server, the files are encoded as `Strings` and stored in a single class file.

This code sample uses packages from the JavaCup winner at JavaOne 1996, which contains the `XImageSource` and `XpmParser` classes. These classes provide all you need to read a standard XPM file. You can see these files at [SunSite](#).

For the initial encoding process, there are a number of graphics tools you can use to create XPM files. On Solaris you can use `ImageTool` or a variety of other [GNU image packages](#). Go to the [Download.com](#) web site to get the encoding software for Windows platforms.

The following code excerpted from the [MyApplet](#) sample class loads the images. You can see the coded `String` form for the images in the [XPM definition](#) of the images.



The `Toolkit` class creates an `Image` object for each image from the XPM Image Source object.

```
Toolkit kit = Toolkit.getDefaultToolkit();
Image image;
image = kit.createImage (new XImageSource (_reply));
image = kit.createImage (new XImageSource (_post));
image = kit.createImage (new XImageSource (_reload));
image = kit.createImage (new XImageSource (_catchup));
image = kit.createImage (new XImageSource (_back10));
image = kit.createImage (new XImageSource (_reset));
image = kit.createImage (new XImageSource (_faq));
```

The alternative technique below uses GIF files. It requires a request back to the web server for each image loaded.


```

Image image;
image = getImage ("reply.gif");
image = getImage ("post.gif");
image = getImage ("reload.gif");
image = getImage ("catchup.gif");
image = getImage ("back10.gif");
image = getImage ("reset.gif");
image = getImage ("faq.gif");

```

This technique reduces network traffic because all images are available in a single class file.

Using XPM encoded images makes the class size larger, but the number of network requests fewer.

Making the XPM image definitions part of your applet class file, makes the image loading process part of the regular loading of the applet class file with no extra classes.

Once loaded, you can use the images to create buttons or other user interface components. This next code segment shows how to use the images with the `javax.swing.JButton` class.

```

ImageIcon icon = new ImageIcon (
    kit.createImage (
        new XImageSource (_reply)));
JButton button = new JButton (icon, "Reply");

```

Using JAR Files

When an applet consists of more than one file, you can improve download performance with Java ARchive (JAR) files. A JAR file contains all of an applet's related files in one single file for a faster download. Much of the time saved comes from reducing the number of HTTP connections the browser must make.

Chapter 9: Deploying Your Application has information on creating and signing JAR files.

The HTML code below uses the `CODE` tag to specify the executable for the `MyApplet` applet, and the `ARCHIVE` tag to specify the JAR file that contains all of `MyApplet`'s related files. The executable specified by the `CODE` tag is sometimes called the `code base`.

For security reasons the JAR files listed by the `archive` parameter must be in the same directory or a sub-directory as the applets `codebase`. If no `codebase` parameter is supplied the directory from

where the applet was loaded is used as the `codebase`.

The following example specifies `jarfile` as the JAR file that contains the related files for the `MyApplet.class` executable.

```
<APPLET CODE="MyApplet.class" ARCHIVE="jarfile" WIDTH="100"
HEIGHT="200"> </APPLET>
```

If the applet download uses multiple JAR files as shown in the next HTML segment, the `ClassLoader` loads each JAR file when the applet starts. So, if your applet uses some resource files infrequently, the JAR file containing those infrequently used files is downloaded, regardless of whether the resources are actually used during that session or not.

```
<APPLET CODE="MyApplet.class" ARCHIVE="jarfile1, jarfile2"
WIDTH="100" HEIGHT="200"> </APPLET>
```

To improve performance when an applet has infrequently used files, put the frequently used files into the JAR file and the infrequently used files into the applet class directory. Infrequently used files are then located and downloaded by the browser only when needed.

Thread Pooling

The Java Developer ConnectionSM (JDC) applet servers and the Java Web ServerTM make extensive use of thread pooling to improve performance. Thread pooling is creating a ready supply of sleeping threads at the beginning of execution. Because the thread startup process is expensive in terms of system resources, thread pooling makes the startup process a little slower, but improves runtime performance because sleeping (or suspended) threads are awakened only when they are needed to perform new tasks.

This code sample taken from the [Pool.java](#) class shows one way to implement thread pooling. In the pool's constructor (shown below), the `WorkerThreads` are initialized and started. The call to the `start` method executes the `run` method of the `WorkerThread`, and the call to `wait` in the `run` method suspends the `Thread` while the `Thread` waits for work to arrive. The last line of the constructor pushes the sleeping `Thread` onto the stack.

```
public Pool (int max, Class workerClass)
    throws Exception {
    _max = max;
```

```

    _waiting = new Stack();
    _workerClass = workerClass;
    Worker worker;
    WorkerThread w;
    for ( int i = 0; i < _max; i++ ) {
        worker = (Worker)_workerClass.newInstance();
        w = new WorkerThread ("Worker#" + i, worker);
        w.start();
        _waiting.push (w);
    }
}

```

Besides the `run` method, the `WorkerThread` class has a `wake` method. When work comes in, the `wake` method is called, which assigns the data and notifies the sleeping `WorkerThread` (the one initialized by the `Pool`) to resume running. The `wake` method's call to `notify` causes the blocked `WorkerThread` to fall out of its wait state, and the `run` method of the [HttpServerWorker](#) class is executed. Once the work is done, the `WorkerThread` is either put back onto the `Stack` (assuming the `Thread Pool` is not full) or terminates.

```

synchronized void wake (Object data) {
    _data = data;
    notify();
}

synchronized public void run(){
    boolean stop = false;
    while (!stop){
        if ( _data == null ){
            try{
                wait();
            } catch (InterruptedException e){
                e.printStackTrace();
            }
            continue;
        }
        if ( _data != null ){
            _worker.run(_data);
        }

        _data = null;
        stop = !(_push (this));
    }
}

```

At its highest level, incoming work is handled by the `performWork` method in the `Pool` class (shown below). As work comes in, an existing `WorkerThread` is popped off of the `Stack` (or a new one is

created if the `Pool` is empty). The sleeping `WorkerThread` is then activated by a call to its `wake` method.

```
public void performWork (Object data)
    throws InstantiationException{
    WorkerThread w = null;
    synchronized (_waiting){
        if ( _waiting.empty() ){
            try{
                w = new WorkerThread ("additional worker",
                    (Worker)_workerClass.newInstance());
                w.start();
            } catch (Exception e){
                throw new InstantiationException (
                    "Problem creating
                    instance of Worker.class: "
                    + e.getMessage());
            }
        } else{
            w = (WorkerThread)_waiting.pop();
        }
    }
    w.wake (data);
}
```

The [HttpServer.java](#) class constructor creates a new `Pool` instance to service [HttpServerWorker](#) instances. `HttpServerWorker` instances are created and stored as part of the `WorkerThread` data. When a `WorkerThread` is activated by a call to its `wake` method, the `HttpServerWorker` instance is invoked by way of its `run` method.

```
try{
    _pool = new Pool (poolSize,
        HttpServerWorker.class);
} catch (Exception e){
    e.printStackTrace();
    throw new InternalError (e.getMessage());
}
```

This next code is in the `run` method of the [HttpServer.java](#) class. Every time a request comes in, the data is initialized and the `Thread` starts work.

Note: If creating a new `Hashtable` for each `WorkerThread` presents too much overhead, just modify the code so it does not use the `Worker` abstraction.

```
try{
    Socket s = _serverSocket.accept();
    Hashtable data = new Hashtable();
    data.put ("Socket", s);
    data.put ("HttpServer", this);
    _pool.performWork (data);
} catch (Exception e){
    e.printStackTrace();
}
```

Thread pooling is an effective performance-tuning technique that puts the expensive thread startup process at the startup of an application. This way, the negative impact on performance occurs once at program startup where it is least likely to be noticed.

The code examples for the thread pooling example are taken from a multi-threaded firewall proxy example. The other files required for this example are as follows:

[Worker.java](#)

[HttpListener.java](#)

[HttpServerListener.java](#)

[HttpClient.java](#)

[\[TOP\]](#)

Printable Page 

[This page was updated: 4-Jun-2001]

[Products & APIs](#) | [Developer Connection](#) | [Docs & Training](#) | [Online Support](#)
[Community Discussion](#) | [Industry News](#) | [Solutions Marketplace](#) | [Case Studies](#)

[Glossary](#) | [Feedback](#) | [A-Z Index](#)

For more information on Java technology
and other software from Sun Microsystems, call:
(800) 786-7638
Outside the U.S. and Canada, dial your country's
[AT&T Direct Access Number](#) first.



Copyright © 1995-2001 [Sun Microsystems, Inc.](#)
All Rights Reserved. [Terms of Use](#). [Privacy Policy](#).



[Products & APIs](#)
[Developer Connection](#)
[Docs & Training](#)
[Online Support](#)
[Community Discussion](#)
[Industry News](#)
[Solutions Marketplace](#)
[Case Studies](#)

[Training Index](#)

Writing Advanced Applications

Chapter 8 Continued: Connection Pooling

[\[<<BACK\]](#)
[\[CONTENTS\]](#)
[\[NEXT>>\]](#)

Printable Page 

■ Requires login

Early Access ■
[Downloads](#)

Bug Database ■
[Submit a Bug](#)
[View Database](#)

Newsletters
[Back Issues](#)
[Subscribe](#)

Learning Centers
[Articles](#)
[Bookshelf](#)
[Code Samples](#)
[New to Java](#)
[Question of the Week](#)
[Quizzes](#)
[Tech Tips](#)
[Tutorials](#)

Forums

If you have used a SQL or other similar tool to connect to a database and act on the data, you probably know that getting the connection and logging in is the part that takes the most time. An application can easily spend several seconds every time it needs to establish a connection.

In releases prior to JDBC™ 2.0 every database session requires a new connection and login even if the previous connection and login used the same table and user account. If you are using a JDBC release prior to 2.0 and want to improve performance, you can cache JDBC connections instead.

Cached connections are kept in a runtime object pool and can be used and reused as needed by the application. One way to implement the object pool is to make a simple hashtable of connection objects. However, a more flexible way to do it is to write a wrapper JDBC `Driver` that is an intermediary between the client application and database.

The wrapper approach works particularly well in an Enterprise Bean that uses Bean-managed persistence for two reasons: 1) Only one `Driver` class is loaded per Bean, and 2) specific connection details are handled outside the Bean.

This section explains how to write a wrapper JDBC `Driver` class.

[Wrapper Classes](#)
[Connection Driver](#)
[Connection Pool](#)
[Deadlocks and Hangs](#)
[Closing Connections](#)
[Example Application](#)

Technology Centers Wrapper Classes

The wrapper JDBC Driver created for this examples consists of the following three classes:

```
JDCConnectionDriver
JDCConnectionPool
JDCConnection
```

Connection Driver

The [JDCConnectionDriver.java](#) class implements the `java.sql.Driver` interface, which provides methods to load drivers and create new database connections.

A `JDCConnectionManager` object is created by the application seeking a database connection. The application provides the database Uniform Resource Locator (URL) for the database, login user ID, and login password.

The `JDCConnectionManager` constructor does the following:

Registers the `JDCConnectionManager` object with the `DriverManager`.

Loads the `Driver` class passed to the constructor by the calling program.

Initializes a `JDCConnectionPool` object for the connections with the database URL, login user ID, and login password passed to the constructor by the calling program.

```
public JDCConnectionDriver(String driver,
                          String url,
                          String user,
                          String password)
    throws ClassNotFoundException,
           InstantiationException,
           IllegalAccessException,
           SQLException {

    DriverManager.registerDriver(this);
    Class.forName(driver).newInstance();
    pool = new JDCConnectionPool(url, user, password);
}
```

When the calling program needs a database connection, it calls the `JDCConnectionDriver.connect` method, which in turn, calls the `JDCConnectionPool.getConnection` method.

Connection Pool

The [JDCCConnectionPool.java](#) class makes connections available to calling program in its `getConnection` method. This method searches for an available connection in the connection pool. If no connection is available from the pool, a new connection is created. If a connection is available from the pool, the `getConnection` method leases the connection and returns it to the calling program.

```
public synchronized Connection getConnection()
    throws SQLException {

    JDCCConnection c;
    for(int i = 0; i < connections.size(); i++) {
        c = (JDCCConnection)connections.elementAt(i);
        if (c.lease()) {
            return c;
        }
    }

    Connection conn = DriverManager.getConnection(
        url, user, password);
    c = new JDCCConnection(conn, this);
    c.lease();
    connections.addElement(c);
    return c;
}
```

The [JDCCConnection.java](#) class represents a JDBC connection in the connection pool, and is essentially a wrapper around a real JDBC connection. The `JDCCConnection` object maintains a state flag to indicate if the connection is in use and the time the connection was taken from the pool. This time is used by the `ConnectionReaper.java` class to identify hanging connections.

Deadlocks and Hangs

While many client and server databases have graceful ways to handle deadlocks and hangs so you do not have to write code to handle these situations, many of the newer, lightweight distributed databases are not so well equipped. The connection pool class provides a dead connection reaper to handle such situations.

The [ConnectionReaper](#) class decides a connection is dead if the following conditions are met.

- The connection is flagged as being in use.

- The connection is older than a preset connection time out.

- The connection fails a validation check.

The validation check runs a simple SQL query over the connection

to see if it throws an exception. In this example, the validation method requests the high-level description of the database tables. If a connection fails the validation test, it is closed, a new connection is initiated to the database, and added to the connection pool.

```
public boolean validate() {
    try {
        conn.getMetaData();
    } catch (Exception e) {
        return false;
    }
    return true;
}
```

Closing Connections

The connection is returned to the connection pool when the calling program calls the `JDBCConnection.close` method in its `finally` clause.

```
public void close() throws SQLException {
    pool.returnConnection(this);
}
```

Example Application

You use a connection pool in an application in a similar way to how you would use any other JDBC driver. Here is the code for a Bean-managed [RegistrationBean](#). This `RegistrationBean` is adapted from the auction house Enterprise JavaBeans™ example described in Chapters 1 - 3.

When the first `RegistrationBean` object is created, it creates one static instance of the `JDBCConnectionDriver` class. This static driver object registers itself with the `DriverManager` in the `JDBCConnectionDriver` constructor making it available for connection requests to all `RegistrationBean` objects created by the client application.

Passing the URL as `jdbc:jdc:jdcpool` in the `getConnection` method, lets the `DriverManager` match the `getConnection` request to the registered driver. The `DriverManager` uses simple String matching to find an available driver that can handle URLs in that format.

```
public class RegistrationBean implements EntityBean{

    private transient EntityContext ctx;
    public String theuser, password;
    public String creditcard, emailaddress;
    public double balance;
```

```
//Static class instantiation
static {
    try{
        new pool.JDCCConnectionDriver(
            "COM.cloudscape.core.JDBCdriver",
            "jdbc:cloudscape:ejbdemo",
            "none", "none");
    }catch(Exception e){}
}

public Connection getConnection()
    throws SQLException{
    return DriverManager.getConnection(
        "jdbc:jdc:jdcpool");
}
}
```

[\[TOP\]](#)

Printable Page 

[This page was updated: 4-Jun-2001]

[Products & APIs](#) | [Developer Connection](#) | [Docs & Training](#) | [Online Support](#)
[Community Discussion](#) | [Industry News](#) | [Solutions Marketplace](#) | [Case Studies](#)

[Glossary](#) | [Feedback](#) | [A-Z Index](#)

For more information on Java technology
and other software from Sun Microsystems, call:
(800) 786-7638
Outside the U.S. and Canada, dial your country's
[AT&T Direct Access Number](#) first.



Copyright © 1995-2001 [Sun Microsystems, Inc.](#)
All Rights Reserved. [Terms of Use](#). [Privacy Policy](#).



- Products & APIs
- Developer Connection
- Docs & Training
- Online Support
- Community Discussion
- Industry News
- Solutions Marketplace
- Case Studies

[Training Index](#)

Writing Advanced Applications

Chapter 8 Continued: Performance Features and Tools

[<<BACK] [CONTENTS] [NEXT>>]

Printable Page 

The new Java™ Virtual Machines (VMs) have features to increase performance, and you can use a number of tools to increase application performance or reduce the size of generated class files. Such features and tools improve the performance of your application with little or no change required to your application.

■ Requires login

Early Access ■
Downloads

- [Java VM Features](#)
- [Just-In-Time Compilers](#)
- [Third-Party Tools](#)

Bug Database ■
Submit a Bug
View Database

Newsletters
Back Issues
Subscribe

Learning Centers
Articles
Bookshelf
Code Samples
New to Java
Question of the Week
Quizzes
Tech Tips
Tutorials

Forums

Java VM Features

The Java® 2 Platform release has introduced many performance improvements over previous releases, including faster memory allocation, reduction of class sizes, improved garbage collection, streamlined monitors and a built-in JIT as standard. When using the new Java 2 VM straight out of the box you will see an improvement, however by understanding how the speed-ups work you can tune your application to squeeze out every last bit of performance.

Method Inlining

The Java 2 release of the Java VM automatically inlines simple methods at runtime. In an un-optimized Java VM, every time a new method is called, a new stack frame is created. The creation of a new stack frame requires additional resources as well as some re-mapping of the stack, the end result is that creating new stack frames incurs a small overhead.

Method inlining increases performance by reducing the number of method calls your program makes. The Java VM inlining code

Technology Centers inlines methods that return constants or only access internal fields. To take advantage of method inlining you can do one of two things. You can either make a method look attractive to the VM to inline or manually inline a method if it doesn't break your object model. Manual inlining in this context means simply moving the code from a method into the method that is calling it.

Automatic VM inlining is illustrated using the following small example:

```
public class InlineMe {
    int counter=0;

    public void method1() {
        for(int i=0;i<1000;i++)
            addCount();
        System.out.println("counter="+counter);
    }

    public int addCount() {
        counter=counter+1;
        return counter;
    }

    public static void main(String args[]) {
        InlineMe im=new InlineMe();
        im.method1();
    }
}
```

In the current state the `addCount` method doesn't look very attractive to the inline detector in the VM because the `addCount` method returns a value. To find out if this method is inlined, run the compiled example with profiling enabled:

```
java -Xrunhprof:cpu=times InlineMe
```

This generates a `java.hprof.txt` output file. The top ten methods will look similar to this:

```
CPU TIME (ms) BEGIN (total = 510)
                Thu Jan 28 16:56:15 1999
rank self accum  count trace method
 1 5.88% 5.88%   1 25 java/lang/Character.
                <clinit>
 2 3.92% 9.80% 5808 13 java/lang/String.charAt
 3 3.92% 13.73%   1 33 sun/misc/
                Launcher$AppClassLoader.
                getPermissions
 4 3.92% 17.65%   3 31 sun/misc/
                URLClassPath.getLoader
```

```
5 1.96% 19.61% 1 39 java/net/  
    URLClassLoader.access$1  
6 1.96% 21.57% 1000 46 InlineMe.addCount  
7 1.96% 23.53% 1 21 sun/io/  
    Converters.newConverter  
8 1.96% 25.49% 1 17 sun/misc/  
    Launcher$ExtClassLoader.  
    getExtDirs  
9 1.96% 27.45% 1 49 java/util/Stack.peek  
10 1.96% 29.41% 1 24 sun/misc/Launcher.<init>
```

If you change the `addCount` method to no longer return a value, the VM will inline it for you at runtime. To make the code inline friendly replace the `addCount` method with the following:

```
public void addCount() {  
    counter=counter+1;  
}
```

And run the profiler again:

```
java -Xrunhprof:cpu=times InlineMe
```

This time the `java.hprof.txt` output should look different. The `addCount` method has gone. It has been inlined!

```
CPU TIME (ms) BEGIN (total = 560)  
    Thu Jan 28 16:57:02 1999  
rank self accum count trace method  
1 5.36% 5.36% 1 27 java/lang/  
    Character.<clinit>  
2 3.57% 8.93% 1 23 java/lang/  
    System.initializeSystemClass  
3 3.57% 12.50% 2 47 java/io/PrintStream.<init>  
4 3.57% 16.07% 5808 15 java/lang/String.charAt  
5 3.57% 19.64% 1 42 sun/net/www/protocol/file/  
    Handler.openConnection  
6 1.79% 21.43% 2 21 java/io/InputStreamReader.fill  
7 1.79% 23.21% 1 54 java/lang/Thread.<init>  
8 1.79% 25.00% 1 39 java/io/PrintStream.write  
9 1.79% 26.79% 1 40 java/util/jar/  
    JarFile.getJarEntry  
10 1.79% 28.57% 1 38 java/lang/Class.forName0
```

Streamlined synchronization

Synchronized methods and objects have until Java 2 always incurred an additional performance hit as the mechanism used to

implement the locking of this code used a global monitor registry which was only single threaded in some areas such as searching for existing monitors. In the Java 2 release, each thread has a monitor registry and so many of the existing bottlenecks have been removed.

If you have previously used other locking mechanisms because of the performance hit with synchronized methods it is now worthwhile re-visiting this code and incorporating the new Java 2 streamlined locks.

In the following example which is creating monitors for the synchronized block you can achieve a 40% speed up. Time taken was 14ms using JDK 1.1.7 and only 10ms with Java 2 on a Sun Ultra 1.

```
class MyLock {

    static Integer count=new Integer(5);
    int test=0;

    public void letslock() {
        synchronized(count) {
            test++;
        }
    }
}

public class LockTest {

    public static void main(String args[]) {

        MyLock ml=new MyLock();
        long time = System.currentTimeMillis();

        for(int i=0;i<5000;i++ ) {
            ml.letslock();
        }
        System.out.println("Time taken="+
            (System.currentTimeMillis()-time));
    }
}
```

Java Hotspot

The Java HotSpot™ VM is Sun Microsystem's next-generation virtual machine implementation. The Java HotSpot VM adheres to the same specification as the Java 2 VM, and runs the same byte codes, but it has been re-engineered to leverage new technologies like adaptive optimization and improved garbage collection models to dramatically improve the speed of the Java VM.

Adaptive optimization

The Java Hotspot does not include a plug-in JIT compiler but instead compiles and inline methods that appear it has determined as being the most used in the application. This means that on the first pass through the Java bytecodes are interpreted as if you did not have a JIT compiler present. If the code then appears as being a hotspot in your application the hotspot compiler will compile the bytecodes into native code which is then stored in a cache and inline methods at the same time. See the inlining section for details on the advantages to inlining code.

One advantage to selective compilation over a JIT compiler is that the byte compiler can spend more time generating highly optimized for the areas that would benefit from the optimization most. The compiler can also avoid compiling code that may be best run in interpreted mode.

Earlier versions of the Java HotSpot VM were not able to optimize code that was not currently in use. The downside to this is if the application was in a huge busy loop the optimizer would not be able to compile the code for area until the loop had finished. Later Java Hotspot VM releases use on-stack replacement, meaning that code can be compiled into native code even if it is in use by the interpreter.

Improved Garbage Collection

The garbage collector used in the Java HotSpot VM introduces several improvements over existing garbage collectors. The first is that the garbage collector is termed a fully accurate collector. What this means is that the garbage collector knows exactly what is an object reference and what is just data. The use of direct references to objects on the heap in a Java HotSpot VM instead of using object handles. This increased knowledge means that memory fragmentation can be reduced which results in a more compact memory footprint.

The second improvement is in the use of generational copying. Java creates a large number of objects on the heap and often these objects are short lived. By placing newly created objects in a memory bucket, waiting for the bucket to fill up and then only copy the remaining live objects to a new area the block of memory that the bucket used can be freed in one block. This means that the VM does not have to search for a hole to fit each new object in the heap and means that smaller sections of memory need to be manipulated at a time.

For older objects the garbage collector makes a sweep through the

heap and compacts holes from dead objects directly, removing the need for a free list used in earlier garbage collection algorithms.

The third area of improvement is to remove the perception of garbage collection pauses by staggering the compaction of large free object spaces into smaller groups and compacting them incrementally.

Fast Thread Synchronization

The Java HotSpot VM also improves existing synchronized code. Synchronized methods and code blocks have always had a performance overhead when run in a Java VM. The Java HotSpot implements the monitor entry and exit synchronization points itself and does not depend on the local OS to provide this synchronization. This results in a large speed improvement especially to often heavily synchronized GUI applications.

Just-In-Time Compilers

The simplest tool used to increase the performance of your application is the Just-In-Time (JIT) compiler. A JIT is a code generator that converts Java bytecode into native machine code. Java programs invoked with a JIT generally run much faster than when the bytecode is executed by the interpreter. The Java Hotspot VM removes the need for a JIT compiler in most cases however you may still find the JIT compiler being used in earlier releases.

The JIT compiler was first made available as a performance update in the Java Development Kit (JDK™) 1.1.6 software release and is now a standard tool invoked whenever you use the `java` interpreter command in the Java 2 platform release. You can disable the JIT compiler using the `-Djava.compiler=NONE` option to the Java VM. This is covered in more detail at the end of the JIT section.

How do JIT Compilers work?

JIT compilers are supplied as standalone platform-dependent native libraries. If the JIT Compiler library exists, the Java VM initializes Java Native Interface (JNI) native code hooks to call JIT functions available in that library instead of the equivalent function in the interpreter.

The `java.lang.Compiler` class is used to load the native library and start the initialization inside the JIT compiler.

When the Java VM invokes a Java method, it uses an invoker method as specified in the method block of the loaded class object. The Java VM has several invoker methods, for example, a different invoker is used if the method is synchronized or if it is a native method.

The JIT compiler uses its own invoker. Sun production releases check the method access bit for value `ACC_MACHINE_COMPILED` to notify the interpreter that the code for this method has already been compiled and stored in the loaded class.

When does the code become JIT compiled code?

When a method is called the first time the JIT compiler compiles the method block into native code for this method and stored that in the code block for that method.

Once the code has been compiled the `ACC_MACHINE_COMPILED` bit, which is used on the Sun platform, is set.

How can I see what the JIT compiler is doing?

The `_JIT_ARGS` environment variable allows simple control of the Sun Solaris JIT compiler. Two useful values are `trace` and `exclude(list)`. To exclude the methods from the `InlineMe` example and show a trace set `_JIT_ARGS` as follows:

Unix:

```
export _JIT_ARGS="trace exclude(InlineMe.addCount
                InlineMe.method1)"
```

```
$ java InlineMe
```

```
Initializing the JIT library ...
```

```
DYNAMICALLY COMPILING java/lang/System.getProperty
                mb= 0x63e74
```

```
DYNAMICALLY COMPILING java/util/Properties.getProperty
                mb= 0x6de74
```

```
DYNAMICALLY COMPILING java/util/Hashtable.get
                mb= 0x714ec
```

```
DYNAMICALLY COMPILING java/lang/String.hashCode
                mb= 0x44aec
```

```
DYNAMICALLY COMPILING java/lang/String.equals
                mb= 0x447f8
```

```
DYNAMICALLY COMPILING java/lang/String.valueOf
                mb= 0x454c4
```

```
DYNAMICALLY COMPILING java/lang/String.toString
                mb= 0x451d0
```

```
DYNAMICALLY COMPILING java/lang/StringBuffer.<init>
                mb= 0x7d690
```

<<<< Inlined java/lang/String.length (4)

Notice that inlined methods such as `String.length` are exempt. The `String.length` is also a special method as it is normally compiled into an internal shortcut bytecode by the Java Interpreter. When using the JIT compiler these optimizations provided by the Java Interpreter are disabled to enable the JIT compiler to understand which method is being called.

How to use the JIT to your advantage

The first thing to remember is that the JIT compiler achieves most of its speed improvements the second time it calls a method. The JIT compiler does compile the whole method instead of interpreting it line by line which can also be a performance gain for when running an application with the JIT enabled. This means that if code is only called once you will not see a significant performance gain. The JIT compiler also ignores class constructors so if possible keep constructor code to a minimum.

The JIT compiler also achieves a minor performance gain by not pre-checking certain Java boundary conditions such as `Null` pointer or array out of bounds exceptions. The only way the JIT compiler knows it has a null pointer exception is by a signal raised by the operating system. Because the signal comes from the operating system and not the Java VM, your program takes a performance hit. To ensure the best performance when running an application with the JIT, make sure your code is very clean with no errors like `Null` pointer or array out of bounds exceptions.

You might want to disable the JIT compiler if you are running the Java VM in remote debug mode, or if you want to see source line numbers instead of the label `(Compiled Code)` in your Java stack traces. To disable the JIT compiler, supply a blank or invalid name for the name of the JIT compiler when you invoke the interpreter command. The following examples show the `javac` command to compile the source code into bytecodes, and two forms of the `java` command to invoke the interpreter without the JIT compiler.

```
javac MyClass.java
java -Djava.compiler=NONE MyClass
```

or

```
javac MyClass.java
java -Djava.compiler="" MyClass
```

Third-Party Tools

Some of the other tools available include those that reduce the size of the generated Java class files. The Java class file contains an area called a constant pool. The constant pool keeps a list of strings and other information for the class file in one place for reference. One of the pieces of information available in the constant pool are the method and field name.

The class file refers to a field in the class as a reference to an entry in the constant pool. This means that as long as the references stay the same, it does not matter what the values stored in the constant pool are. This knowledge is exploited by several tools that rewrite the names of the field and methods in the constant pool into shortened names. This technique can reduce the class file by a significant percentage with the benefit that a smaller class file means a shorter network download.

[\[TOP\]](#)

Printable Page 

[This page was updated: 4-Jun-2001]

[Products & APIs](#) | [Developer Connection](#) | [Docs & Training](#) | [Online Support](#)
[Community Discussion](#) | [Industry News](#) | [Solutions Marketplace](#) | [Case Studies](#)

[Glossary](#) | [Feedback](#) | [A-Z Index](#)

For more information on Java technology
and other software from Sun Microsystems, call:
(800) 786-7638
Outside the U.S. and Canada, dial your country's
[AT&T Direct Access Number](#) first.



Copyright © 1995-2001 [Sun Microsystems, Inc.](#)
All Rights Reserved. [Terms of Use](#). [Privacy Policy](#).



[Products & APIs](#)
[Developer Connection](#)
[Docs & Training](#)
[Online Support](#)
[Community Discussion](#)
[Industry News](#)
[Solutions Marketplace](#)
[Case Studies](#)

[Training Index](#)

Writing Advanced Applications

Chapter 8 Continued: Performance Analysis

[\[<<BACK\]](#) [\[CONTENTS\]](#) [\[NEXT>>\]](#)

[Printable Page](#) 

■ Requires login

[Early Access](#) ■
[Downloads](#)

[Bug Database](#) ■
[Submit a Bug](#)
[View Database](#)

[Newsletters](#)
[Back Issues](#)
[Subscribe](#)

[Learning Centers](#)
[Articles](#)
[Bookshelf](#)
[Code Samples](#)
[New to Java](#)
[Question of the Week](#)
[Quizzes](#)
[Tech Tips](#)
[Tutorials](#)

[Forums](#)

Another way to improve performance is with performance analysis. Performance analysis is looking at program execution to pinpoint where bottlenecks or other performance problems such as memory leaks might occur. Once you know where potential trouble spots are, you can change your code to remove or reduce their impact.

[Profiling](#)

[Analyze a Program](#)

[Operating System Performance Tools](#)

Profiling

The Java™ Virtual Machines (VMs) have had the ability to provide simple profile reports since Java Development Kit (JDK™) 1.0.2. However, the information they provided was limited to a sorted list of method calls a program had called.

The Java® 2 platform software provides much better profiling capabilities than previously available and analysis of this generated data has been made easier by the emergence of a Heap Analysis Tool (HAT). The heap analysis tool, as its name implies, lets you analyze profile reports of the heap. The heap is a block of memory the Java VM uses when it is running. The heap analysis tool lets you generate reports on objects that were used to run your application. Not only can you get a listing of the most frequently called methods and the memory used in calling those methods, but you can also track down memory leaks. Memory leaks can have a significant impact on performance.

Download the Heap Analysis tool:

[binary](#)

Analyze a Program

To analyze the `TableExample3` program included in the `demo/jfc/Table` directory in the Java 2 platform download, you need to generate a profile report. The simplest report to generate is a text profile. To generate a text profile, run the application with the `-Xhprof` parameter. In the final release of the Java 2 platform software, this option was renamed `-Xrunhprof`. To see a list of the currently available options run the command

```
java -Xrunhprof:help
Hprof usage: -Xrunhprof[:help][<option>=<value>, ...]
```

Option Name and Value	Description	Default
-----	-----	-----
heap=dump sites all	heap profiling	all
cpu=samples times old	CPU usage	off
monitor=y n	monitor contention	n
format=a b	ascii or binary output	a
file=<file>	write data to file	java.hprof(.txt for ascii)
net=<host>:<port>	send data over a socket	write to file
depth=<size>	stack trace depth	4
cutoff=<value>	output cutoff point	0.0001
lineno=y n	line number in traces	y
thread=y n	thread in traces?	n
doe=y n	dump on exit?	y

```
Example: java -Xrunhprof:cpu=samples,file=log.txt,
          depth=3 FooClass
```

The following invocation creates a text output file that you can view without the heap analysis tool called `java.hprof.txt` when the program generates a stack trace or exits. A different invocation is used to create a binary file to use with the heap analysis tool.

```
java -Xrunhprof TableExample3

d:\jdk12\demo\jfc\Table> java -Xrunhprof TableExample3
Dumping Java heap ... allocation sites ... done.
```

The `profile` option literally logs every object created on the heap, so even just starting and stopping the small `TableExample3` program results in a four megabyte report file. Although the heap analysis tool uses a binary version of this file and provides a summary, there are some quick and easy things you can learn from the text file without using the heap analysis tool.

Note: To list all available options, use

```
java -Xrunhprof:help
```

View the Text File

Choose an editor that can handle large files and go to the end of this file. There could be hundreds of thousands of lines, so use a shortcut instead of scrolling, or search for the words `SITES BEGIN`. You should see a list of lines that start with an increasing rank number followed by two percentage numbers. The first entry in this list, should look similar to the example below:

```
SITES BEGIN (ordered by live bytes)
                Sun Dec 20 16:33:28 1998

                percent      live      alloc'ed stackclass
rankself accum bytes objsbytes objstracename
1  55.86% 55.86% 8265165    8265165    3981 [S
```

The `[S` notation at the end of the last line above indicates the first entry is an array of the `short`, a primitive type. This is expected with Swing or Abstract Window Toolkit (AWT) applications. The five count under the `objs` header mean there are currently five of these arrays, there have only been five in the lifetime of this application, and they take up 826516 bytes. The reference key to this object is the value listed under `stack trace`. To find where this object was created in this example, search for `TRACE 3981`. You will see the following:

```
TRACE 3981:
java/awt/image/DataBufferUShort.<init>(
                                DataBufferUShort.java:50)
java/awt/image/Raster.createPackedRaster(
                                Raster.java:400)
java/awt/image/DirectColorModel.
  createCompatibleWritableRaster(
                                DirectColorModel.java:641)
sun/awt/windows/WComponentPeer.createImage(
                                WComponentPeer.java:186)
```

The `TableExample3` code sets a scrollpane that is 700 by 300. When

you look at the source of `Raster.java`, which is in the `src.jar` file, you find these statements at line 400:

```
case DataBuffer.TYPE_USHORT:
    d = new DataBufferUShort(w*h);
    break;
```

The values `w` and `h` are the width and height from the `createImage` call at the start of TRACE 3981. The `DataBufferUShort` constructor creates an array of shorts as follows:

```
data = new short[size];
```

where `size` is `w*h`. So, in theory there should be an entry for an array of 210000 elements. You look for each instantiation of this class by searching for `trace=3981`. One of the five entries will look like this:

```
OBJ 5ca1fc0 (sz=28, trace=3979,
    class=java/awt/image/DataBufferUShort@9a2570)
data 5ca1670
bankdata 5ca1f90
offsets 5ca1340
ARR 5ca1340 (sz=4, trace=3980, nelems=1,
    elem type=int)
ARR 5ca1670 (sz=420004, trace=3981, nelems=210000,
    elem type=short)
ARR 5ca1f90 (sz=12, trace=3982, nelems=1,
    elem type=[S@9a2d90)
[0] 5ca1670
```

You can see that the data value of this raster image references an array `5ca1670` which in turn lists 210000 elements of a short of size 2. This means 420004 bytes of memory are used in this array.

From this data you can conclude that the `TableExample3` program uses nearly 0.5Mb to map each table. If the example application is running on a small memory machine, you should make sure you do not keep unnecessary references to large tables or images that are built by the `createImage` method.

The Heap Analysis Tool

The Heap Analysis tool can analyze the same data for you, but requires a binary report file as input. You can generate a binary report file as follows:

```
java -Xrunhprof:file=TableExample3.hprof,format=b
    TableExample3
```

To generate the binary report, close the `TableExample3` window. The

binary report file `TableExample3.hprof` is created when the program exits. The Heap Analysis tool starts an HTTP Server that analyzes the binary profile file and displays the results in HTML that you can view with a browser.

You can get a copy of the Heap Analysis Tool from the java.sun.com site. Once you install it, you run shell and batch scripts in the installed `bin` directory so you can start the Heap Analysis Tool server as follows:

```
>hat TableExample3.hprof
Started HCODEP server on port 7000
Reading from /tmp/TableExample3.hprof...
Dump file created Tue Jan 05 13:28:59 PST 1999
Snapshot read, resolving...
Resolving 17854 objects...
Chasing references,
    expect 35 dots.....
Eliminating duplicate
    references.....
Snapshot resolved.
Server is ready.
```

The above output tells you an HTTP server is started on port 7000 by default. To view this report enter the url `http://localhost:7000` or `http://your_machine_name:7000` in your web browser. If you have problems starting the server using the scripts, you can alternatively run the application by including the `hat.zip` classes file on your `CLASSPATH` and use the following command:

```
java hat.Main TableExample3.hprof
```

The default report view contains a list of all the classes. At the bottom of this initial page are the following two key report options:

```
Show all members of the rootset
Show instance counts for all classes
```

If you select the `Show all members of the rootset` link, you see a list of the following references because these reference are likely targets for potential memory leaks.

```
Java Static References
Busy Monitor References
JNI Global References
JNI Local References
System Class References
```

What you look for here are instances in the application that have references to objects that have a risk of not being garbage collected. This can sometimes occur in the case of JNI if memory is allocated for an object, the memory is left to the garbage collector

to free up, and the garbage collector does not have the information it needs to do it. In this list of references, you are mainly interested in a large number of references to objects or objects of a large size.

The other key report is the Show instance counts for all classes. This lists the number of calls to a particular method. The `String` and `Character` array objects, `[S` and `[C`, are always going to be high on this list, but some objects are a bit more intriguing. Why are there 323 instances of `java.util.SimpleTimeZone` for example?

```
5109 instances of class java.lang.String
5095 instances of class [C
2210 instances of class java.util.Hashtable$Entry
968 instances of class java.lang.Class
407 instances of class [Ljava.lang.String;
323 instances of class java.util.SimpleTimeZone
305 instances of class
    sun.java2d.loops.GraphicsPrimitiveProxy
304 instances of class java.util.HashMap$Entry
269 instances of class [I
182 instances of class [Ljava.util.Hashtable$Entry;
170 instances of class java.util.Hashtable
138 instances of class java.util.jar.Attributes$Name
131 instances of class java.util.HashMap
131 instances of class [Ljava.util.HashMap$Entry;
130 instances of class [Ljava.lang.Object;
105 instances of class java.util.jar.Attributes
```

To get more information on the `SimpleTimeZone` instances, click on the link (the line beginning with 323). This will list all 323 references and calculate how much memory has been used. In this example, 21964 bytes have been used.

```
Instances of java.util.SimpleTimeZone

class java.util.SimpleTimeZone

java.util.SimpleTimeZone@0x004f48c0 (68 bytes)
java.util.SimpleTimeZone@0x003d5ad8 (68 bytes)
java.util.SimpleTimeZone@0x004fae88 (68 bytes)
.....
Total of 323 instances occupying 21964 bytes.
```

If you click on one of these `SimpleTimeZone` instances, you see where this object was allocated.

```
Object allocated from:

java.util.TimeZoneData.<clinit>(()V) :
    TimeZone.java line 1222
java.util.TimeZone.getTimeZone((Ljava/lang/String;)
    Ljava/util/TimeZone;) :
```

```

        TimeZone.java line (compiled method)
java.util.TimeZone.getDefault(
    ()Ljava/util/TimeZone;) :
        TimeZone.java line (compiled method)
java.text.SimpleDateFormat.initialize(
    (Ljava/util/Locale;)V) :
        SimpleDateFormat.java line (compiled method)

```

In this example the object was allocated from `TimeZone.java`. The source to this file is in the standard `src.jar` file, and on examining this file, you can see that indeed there are nearly 300 of these objects in memory.

```

static SimpleTimeZone zones[] = {
    // The following data is current as of 1998.
    // Total Unix zones: 343
    // Total Java zones: 289
    // Not all Unix zones become Java zones due to
    // duplication and overlap.
    //-----
    new SimpleTimeZone(-11*ONE_HOUR,
        "Pacific/Niue" /*NUT*/),

```

Unfortunately, you have no control over the memory used in this example because it is allocated when the program first requests a default timezone. However, this same technique can be applied to analyzing your own application where you may be able to make some improvements

Where the Application Spends its Time

Again, you can use the `-Xrunhprof` parameter to get information about the time the application spent processing a particular method.

You can use one of two CPU profiling options to achieve this. The first option is `cpu=samples`. This option reports the result of a sampling of the running threads of the Java VM to which a statistical count of the frequency of the occurrence of a particular method is used to find busy sections of the applications. The second option is `cpu=times`, which measures the time taken by individual methods and generates a sorted list ranked as a total percentage of the CPU time taken by the application.

By using the `cpu=times` option, you should see something similar to this at the end of the output file

```

CPU TIME (ms) BEGIN (total = 11080)
                        Fri Jan  8 16:40:59 1999
rank  self   accum  count  trace  method
  1   13.81%  13.81%    1     437  sun/

```

```

    awt/X11GraphicsEnvironment.initDisplay
2   2.35%  16.16%      4   456   java/
    lang/ClassLoader$NativeLibrary.load
3   0.99%  17.15%     46   401   java/
    lang/ClassLoader.findBootstrapClass

```

If you contrast this with the `cpu=samples` output, you see the difference between how often a method appears during the runtime of the application in the samples output compared to how long that method took in the times output.

```

CPU SAMPLES BEGIN (total = 14520)
                               Sat Jan 09 17:14:47 1999
rank  self   accum   count  trace  method
  1   2.93%  2.93%   425    2532  sun/
      awt/windows/WGraphics.W32LockViewResources
  2   1.63%  4.56%   237     763  sun/
      awt/windows/WToolkit.eventLoop
  3   1.35%  5.91%   196    1347  java/
      text/DecimalFormat.<init>

```

The `W32LockView` method, which calls a native windows lock routine, is called 425 times. So when it is sampled it appears in the active runnings threads because it also takes time to complete. In contrast, the `initDisplay` method is only called once, but it is the method that takes the longest time to complete in real time.

Operating System Performance Tools

Sometimes the performance bottleneck occurs at the system or operating system level. This is because Java VM depends on many operating system libraries for functionality such as disk access or networking. However, what occurs in these libraries after the Java VM calls them is beyond the reach of most profiling tools for the Java platform.

Here is a list of tools you can use to analyze performance problems on some common operating systems.

Solaris Platform

System accounting reports, `sar`, reports the activity of the system in terms of disk IO, user program activity, and system level activity. If your application is using excessive amounts of memory, it may require disk swap space, which will show up as high percentages in the `WIO` column. User programs that get stuck in a busy loop show a high percentage in the `user` column:

```

developer$ sar 1 10

```

```
SunOS developer 5.6 Generic_105181-09 sun4u
                                02/05/99
```

```
11:20:29    %usr    %sys    %wio    %idle
11:20:30      30      6      9      55
11:20:31      27      0      3      70
11:20:32      25      1      1      73
11:20:33      25      1      0      74
11:20:34      27      0      1      72
```

The `truss` command traces and records the details of every system call called by the Java VM to the Solaris kernel. A common way to run `truss` is:

```
truss -f -o /tmp/output -p <process id>
```

The `-f` parameter follows any child processes that are created, the `-o` parameter writes the output to the named file, and the `-p` parameter traces an already running program from its process ID. Alternately, you can replace `-p <process id>` with the Java VM, for example:

```
truss -f -o /tmp/output java MyDaemon
```

The `/tmp/output` is used to store the `truss` output, which should look similar to the following:

```
15573:  execve("/usr/local/java/jdk1.2/solaris/
           bin/java", 0xEFFFFFF2DC,
           0xEFFFFFF2E8)  argc                = 4
15573:  open("/dev/zero", O_RDONLY)                = 3
15573:  mmap(0x00000000, 8192,
           PROT_READ|PROT_WRITE|PROT_EXEC,
           MAP_PRIVATE, 3, 0) = 0xEF7C0000
15573:  open("/home/calvin/java/native4/libsocket.so.1",
           O_RDONLY) Err#2 ENOENT
15573:  open("/usr/lib/libsocket.so.1",
           O_RDONLY)                = 4
15573:  fstat(4, 0xEFFFFFFF6C)                    = 0
15573:  mmap(0x00000000, 8192, PROT_READ|PROT_EXEC,
           MAP_SHARED, 4, 0) = 0xEF7B00 00
15573:  mmap(0x00000000, 122880, PROT_READ|PROT_EXEC,
           MAP_PRIVATE, 4, 0) = 0xEF7 80000
15573:  munmap(0xEF78E000, 57344)                  = 0
15573:  mmap(0xEF79C000, 5393,
           PROT_READ|PROT_WRITE|PROT_EXEC,
           MAP_PRIVATE|MAP_FIXED, 4, 49152)
           = 0xEF79C000
15573:  close(4)                                    = 0
```

In the `truss` output, look for files that fail when opened due to access problems, such as error `ENOPERM`, or a missing file error

ENOENT. You can also track data read or written with the `truss` parameters `-rall` to log all data read or `-wall` to log all data written by the program. With these parameters, it is possible to analyze data sent over a network or to a local disk.

Linux Platform

Linux has a trace command called `strace`. It traces systems calls to the underlying Linux kernel. This example traces the Spreadsheet example in the JDK demo directory.

```
$ strace -f -o /tmp/output
                               java sun.applet.AppletViewer
                               example1.html
$ cat /tmp/output

639  execve("/root/java/jdk117_v1at/java/
                               jdk117_v1a/bin/java", ["java",
                               "sun.applet.AppletViewer ",
                               "example1.html"], [/* 21 vars */]) = 0
639  brk(0)                               = 0x809355c
639  open("/etc/ld.so.preload", O_RDONLY)  = -1
                               ENOENT (No such file or directory)
639  open("/etc/ld.so.cache", O_RDONLY)     = 4
639  fstat(4, {st_mode=0, st_size=0, ...})   = 0
639  mmap(0, 14773, PROT_READ, MAP_PRIVATE,
                               4, 0) = 0x4000b000
639  close(4)                                = 0
639  open("/lib/libtermcap.so.2", O_RDONLY) = 4
639  mmap(0, 4096, PROT_READ, MAP_PRIVATE,
                               4, 0) = 0x4000f000
```

To obtain system information similar to the Solaris `sar` command, read the contents of the file `/proc/stat`. The format of this file is described in the `proc` man page. Look at the `cpu` line to get the user and system time.

```
cpu  4827 4 1636 168329
```

In the above example, the `cpu` line indicates 48.27 seconds in user space, 0.04 at nice priority, 16.36 seconds processing system calls, and 168 seconds idle. This is a running total; individual entries for each process are available in `/proc/<process_id>/stat`.

Windows95/98/NT Platforms

There are no standard performance analysis tools included on this platform, but the following tools are available by way of freeware or shareware resources such as <http://www.download.com> .

Runtime memory analysis: Memory meter

Network analysis: Traceplus

[\[TOP\]](#)

Printable Page 

[This page was updated: 4-Jun-2001]

[Products & APIs](#) | [Developer Connection](#) | [Docs & Training](#) | [Online Support](#)
[Community Discussion](#) | [Industry News](#) | [Solutions Marketplace](#) | [Case Studies](#)

[Glossary](#) | [Feedback](#) | [A-Z Index](#)

For more information on Java technology
and other software from Sun Microsystems, call:
(800) 786-7638
Outside the U.S. and Canada, dial your country's
[AT&T Direct Access Number](#) first.



Copyright © 1995-2001 [Sun Microsystems, Inc.](#)
All Rights Reserved. [Terms of Use](#). [Privacy Policy](#).



[Products & APIs](#)
[Developer Connection](#)
[Docs & Training](#)
[Online Support](#)
[Community Discussion](#)
[Industry News](#)
[Solutions Marketplace](#)
[Case Studies](#)

[Training Index](#)

[Printable Page](#) 

■ Requires login

[Early Access](#) ■
[Downloads](#)

[Bug Database](#) ■
[Submit a Bug](#)
[View Database](#)

[Newsletters](#)
[Back Issues](#)
[Subscribe](#)

[Learning Centers](#)
[Articles](#)
[Bookshelf](#)
[Code Samples](#)
[New to Java](#)
[Question of the Week](#)
[Quizzes](#)
[Tech Tips](#)
[Tutorials](#)

[Forums](#)

Writing Advanced Applications

Chapter 8 Continued: Caching Client/Server Applications

[<<BACK] [CONTENTS] [NEXT>>]

Caching is one of the first techniques used to improve the performance of web browsers and web servers. The browser cache makes network lookup operations unnecessary because a recent copy of the file is kept in the local cache, and the web server cache reduces the cost of loading the file from disk for each request. This section explains how you can use caching in a similar way to improve performance in many client/server applications written in the Java™ programming language.

The `java.util.Collections` API available in the Java® 2 Software Development Kit (SDK) software makes implementing a cache simple. This API provides the `HashMap` class, which works well for caching one object, and the `LinkedList` class, which works well in combination with the `HashMap` class for caching many objects.

[Caching One Object](#)

[Caching Many Objects](#)

Caching One Object

A `HashMap` object stores data in key and value pairs. When you put a data value in the `HashMap`, you assign it a key and later use that key to retrieve the data.

A `HashMap` object is very similar to a `Hashtable` and can be used to keep a temporary copy of previously generated results. Objects kept in the `HashMap` cache could, for example, be a list of completed auction results.

In this case, the results of a JDBC query might be requested hundreds of times a second by persons wanting to know who was the highest bidder, but the completed results lists only actually

Technology Centers changes once a minute as each auction completes. You can write your program to retrieve unchanged objects from the results cache instead of querying the database every time and gain a significant performance improvement.

This [code example](#) runs a database query once a minute, and returns cached copies for requests that come between the queries.

```
import java.util.*;
import java.io.*;

class DBCacheRecord {
    Object data;
    long time;

    public DBCacheRecord(Object results, long when) {
        time=when;
        data=results;
    }
    public Object getResults() {
        return data;
    }
    public long getLastModified() {
        return time;
    }
}

public class DBCache {
    Map cache;

    public DBCache() {
        cache = new HashMap();
    }

    public Object getDBData(String dbcommand) {
        if(!cache.containsKey(dbcommand)) {
            synchronized(cache) {
                cache.put(dbcommand, readDBData(dbcommand));
            }
        } else {
            if((new Date().getTime() ) -
                ((DBCacheRecord)cache.get(
                    dbcommand)).getLastModified())>=1000) {
                synchronized(cache) {
                    cache.put(dbcommand, readDBData(dbcommand));
                }
            }
        }
        return ((DBCacheRecord)cache.get(
            dbcommand)).getResults();
    }

    public Object readDBData(String dbcommand) {

        /*Insert your JDBC code here For Example:
        ResultSet results=stmt.executeQuery(dbcommand);
        */

        String results="example results";
        return(new DBCacheRecord(results,new
```



```

        Date().getTime()));
    }

    public static void main(String args[]) {
        DBCache dl=new DBCache();
        for(int i=1;i<=20;i++) {
            dl.getDBData(
                "select count(*) from results where
                TO_DATE(results.completed) <=SYSDATE");
        }
    }
}

```

Caching Many Objects

Sometimes you will want to cache more than one object. For example, you might want to keep the most recently accessed files on a web server in a cache. If you use a `HashMap` object for a purpose like this, it will continue to grow and use a lot of memory.

If your machine has large amounts of memory and only a small number of objects to cache then a growing `HashMap` may not be a problem. However, if you are intending to cache a lot of objects then you may find that keeping only the most recent objects in the cache provides the best use of the machine's memory. You can combine a `HashMap` object with a `LinkedList` to create what is called a Most Recently Used (MRU) cache.

Note: There are other techniques used to constrain cache size besides MRU. MRU is one of the simpler algorithms.

With an MRU cache, you can place a constraint on which objects remain in cache, and thereby, control the size of the cache. There are three main operations that the MRU cache has to perform:

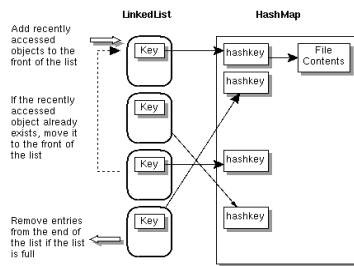
If the cache is not full, new objects not already in the cache are inserted at the head of the list.

If the cache is not full and the object to be inserted already exists in the cache, it is moved to the head of the list.

If the cache is full and a new object is to be inserted, the last object in the cache is removed and the new object is inserted at the head of the list.

This diagram shows how the `LinkedList` and `HashMap` work together to implement the operations described above. A discussion of the

diagram follows.



MRU Cache with LinkedList and HashMap

The `LinkedList` provides the queue mechanism, and the entries in the `LinkedList` contain the key to the data in the `HashMap`. To add a new entry to the front of the list, the `addFirst` method is called.

If the list is already full, the `removeLast` method is called and the data entry is also removed from the `HashMap`.

If an entry was already in the list, it is removed with a call to the `remove` method and inserted at the front of the list with a call to the `addFirst` method.

The Collections API does not implement locking, so if you remove entries from or add entries to `LinkedList` or `HashMap` objects, you need to lock access to these objects. You can also use a `Vector` or `ArrayList` to get the same results as shown in the code below with the `LinkedList`.

This [code example](#) uses an MRU cache to keep a cache of files loaded from disk. When a file is requested, the program checks to see if the file is in the cache. If the file is not in the cache, the program reads the file from disk and places the cache copy at the beginning of the list.

If the file is in cache, the program compares the modification times of the file and cache entry.

If the cache entry time is older, the program reads the file from disk, removes the cache copy, and places a new copy in the cache at the front of the `LinkedList`.

If the file time is older, the program gets the file from the cache and moves the cache copy to the front of the list.

```
import java.util.*;
import java.io.*;

class myFile {
    long lastmodified;
    String contents;

    public myFile(long last, String data) {
        lastmodified=last;
        contents=data;
    }
    public long getLastModified() {
        return lastmodified;
    }
    public String getContents() {
        return contents;
    }
}

public class MRUCache {

    Map cache;
    LinkedList mrulist;
    int cachesize;

    public MRUCache(int max) {
        cache = new HashMap();
        mrulist= new LinkedList();
        cachesize=max;
    }

    public String getFile(String fname) {
        if(!cache.containsKey(fname)) {
            synchronized(cache) {
                if(mrulist.size() >=cachesize) {
                    cache.remove(mrulist.getLast());
                    mrulist.removeLast();
                }
                cache.put(fname, readFile(fname));
                mrulist.addFirst(fname);
            }
        } else {
            if((new File(fname).lastModified())>
                ((myFile)cache.get(fname)).getLastModified()) {
                synchronized(cache) {
                    cache.put(fname, readFile(fname));
                }
            }
        }
    }
}
```

```

        synchronized(cache) {
            mrulist.remove(fname);
            mrulist.addFirst(fname);
        }
    }
    return ((myFile)cache.get(fname)).getContents();
}

public myFile readFile(String name) {
    File f = new File(name);
    StringBuffer filecontents= new StringBuffer();

    try {
        BufferedReader br=new BufferedReader(
                                new FileReader(f));

        String line;

        while((line =br.readLine()) != null) {
            filecontents.append(line);
        }
    } catch (FileNotFoundException fnfe){
        return (null);
    } catch ( IOException ioe) {
        return (null);
    }
    return (new myFile(f.lastModified(),
        filecontents.toString()));
}

public void printList() {
    for(int i=0;i<mrulist.size();i++) {
        System.out.println("item "+i+"="+mrulist.get(i));
    }
}

public static void main(String args[]) {

    // Number of entries in MRU cache is set to 10
    MRUCache h1=new MRUCache(10);
    for(int i=1;i<=20;i++) {
        // files are stored in a subdirectory called data
        h1.getFile("data"+File.separatorChar+i);
    }
    h1.printList();
}
}

```

[\[TOP\]](#)[Printable Page](#) 

[This page was updated: 4-Jun-2001]

[Products & APIs](#) | [Developer Connection](#) | [Docs & Training](#) | [Online Support](#)
[Community Discussion](#) | [Industry News](#) | [Solutions Marketplace](#) | [Case Studies](#)

[Glossary](#) | [Feedback](#) | [A-Z Index](#)

For more information on Java technology
and other software from Sun Microsystems, call:
(800) 786-7638
Outside the U.S. and Canada, dial your country's
[AT&T Direct Access Number](#) first.



Copyright © 1995-2001 [Sun Microsystems, Inc.](#)
All Rights Reserved. [Terms of Use.](#) [Privacy Policy.](#)



[Products & APIs](#)
[Developer Connection](#)
[Docs & Training](#)
[Online Support](#)
[Community Discussion](#)
[Industry News](#)
[Solutions Marketplace](#)
[Case Studies](#)

[Training Index](#)

[Printable Page](#) 

■ Requires login

[Early Access](#) ■
[Downloads](#)

[Bug Database](#) ■
[Submit a Bug](#)
[View Database](#)

[Newsletters](#)
[Back Issues](#)
[Subscribe](#)

[Learning Centers](#)
[Articles](#)
[Bookshelf](#)
[Code Samples](#)
[New to Java](#)
[Question of the Week](#)
[Quizzes](#)
[Tech Tips](#)
[Tutorials](#)

[Forums](#)

Writing Advanced Applications

Chapter 9: Deploying the Auction Application

[<<BACK] [CONTENTS] [NEXT>>]

With the auction application tested, debugged, and tuned, you are ready to deploy it. Deployment involves bundling the application files, moving the application files to their production locations, installing Java Plug-In so auction administrators can run the Administration applet from their browsers, and installing the Administration applet policy file. Java Plug-In is needed because the Administration applet is written with Java Development Kit (JDK™) 1.2 APIs, but the administrators' browsers might run an earlier version of the Java Runtime Environment™ (JRE) software.

This chapter explains how to use the Java Archive (JAR) file format to bundle and deploy the application files, and how to install Java Plug-In and a security policy file for the Solaris™ and Win32 platforms to run the Administration applet.

[Java Archive File Format](#)

[Solaris Platform](#)

[Win32 Platform](#)

In a Rush?

This table links you directly to specific topics.

Topic	Section
JAR File Format	Bundle and Deploy the HTML Files Bundle and Deploy the Enterprise Beans Bundle and Deploy the Administration Applet

Technology Centers

Solaris Platform	Get Downloads Extract Downloaded Files Install Java Plug-In Install Java Plug-In Patches Install Netscape Communicator Check the Installation Convert HTML Files Security Policy File Run the Administration Applet
Win32 Platform	Download and Install Convert HTML Files Security Policy Files Run the Administration Applet

[[TOP](#)]

Printable Page 

[This page was updated: 4-Jun-2001]

[Products & APIs](#) | [Developer Connection](#) | [Docs & Training](#) | [Online Support](#)
[Community Discussion](#) | [Industry News](#) | [Solutions Marketplace](#) | [Case Studies](#)

[Glossary](#) | [Feedback](#) | [A-Z Index](#)

For more information on Java technology
and other software from Sun Microsystems, call:
(800) 786-7638
Outside the U.S. and Canada, dial your country's
[AT&T Direct Access Number](#) first.



Copyright © 1995-2001 [Sun Microsystems, Inc.](#)
All Rights Reserved. [Terms of Use](#). [Privacy Policy](#).



[Products & APIs](#)
[Developer Connection](#)
[Docs & Training](#)
[Online Support](#)
[Community Discussion](#)
[Industry News](#)
[Solutions Marketplace](#)
[Case Studies](#)

[Training Index](#)

Writing Advanced Applications

Chapter 9 Continued: JAR File Format

[\[<<BACK\]](#)
[\[CONTENTS\]](#)
[\[NEXT>>\]](#)

Java™ Archive (JAR) file format is a compression and file packaging format and tool for bundling executable files with other related application files so they can be deployed as a single unit. The auction application, has three units of files to deploy to three different locations.

■ Requires login

[Early Access](#) ■
[Downloads](#)

[Bug Database](#) ■
[Submit a Bug](#)
[View Database](#)

[Newsletters](#)
[Back Issues](#)
[Subscribe](#)

[Learning Centers](#)
[Articles](#)
[Bookshelf](#)
[Code Samples](#)
[New to Java](#)
[Question of the Week](#)
[Quizzes](#)
[Tech Tips](#)
[Tutorials](#)

[Forums](#)

1. The HTML files that make up the auction application user interface deploy to a publicly accessible location under the web server.
2. The Enterprise Beans deploy to an internal location accessible to your production installation of the Enterprise JavaBeans™ server.
3. The Administration applet deploys to an internal location accessible to auction administrators where it is run from their browsers.

This section shows you how to use the `jar` tool to bundle and deploy the application files.

[Bundle and Deploy the HTML Files](#)

[Bundle and Deploy the Enterprise Beans](#)

[Bundle and Deploy the Administration Applet](#)

Bundle and Deploy the HTML Files

Here is the list of HTML files that make up the auction application user interface:

[all.html](#)
[close.html](#)
[details.html](#)
[index.html](#)

Technology Centers

juggler.med.gif
new.html
registration.html
search.html
sell.html

Here is the `jar` command to bundle them. Everything goes on one line. This command is executed in the same directory with the files. If you execute the command from a directory other than where the files are, specify the full or relative pathname as appropriate.

```
jar cvf HTML.jar all.html close.html details.html
      index.html juggler.med.gif new.html
      registration.html search.html sell.html
```

`jar` is the `jar` command. If you type `jar` with no options, you get the following help screen. You can see from the help screen that the `cf` options to the `jar` command mean create a new JAR file named `HTML.jar` and put the list of files that follows into it. The new JAR file is placed in the current directory.

```
kq6py% jar
```

```
Usage: jar {ctxu} [vfmOM] [jar-file] [manifest-file]
        [-C dir] files ...
```

Options:

- c create new archive
- t list table of contents for archive
- x extract named (or all) files from archive
- u update existing archive
- v generate verbose output on standard output
- f specify archive file name
- m include manifest information from specified manifest file
- 0 store only; use no ZIP compression
- M Do not create a manifest file for the entries
- C change to the specified directory and include the following file

If any file is a directory then it is processed recursively. The manifest file name and the archive file name needs to be specified in the same order the 'm' and 'f' flags are specified.

Example 1: to archive two class files into an archive called `classes.jar`:

```
jar cvf classes.jar Foo.class Bar.class
```

Example 2: use an existing manifest file 'mymanifest' and archive all the files in the `foo/` director into 'classes.jar':

```
jar cvfm classes.jar mymanifest -C foo/ .
```

To deploy the HTML files, all you have to do is move the `HTML.jar` file to a publicly accessible directory under the web server and

decompress the JAR file:

```
jar xf HTML.jar
```

Note: If you included a full or relative pathname when you added the files to the JAR file, the files are placed in the same directory structure when they are unpacked.

Bundle and Deploy the Enterprise Beans

Some Enterprise JavaBeans servers create the JAR file for you. However, if yours does not or if you just wonder how it's done, this section describes the steps.

Here are the server-side files you need to deploy the Enterprise Beans. This list is taken from the original auction application described in [Chapter 2: Auction Application Code](#) before any modifications were made to make the Enterprise Beans container managed. Note the inclusion of the deployment descriptor, and the container-generated stub and skel classes.

auction Package

Here are the application files in the `auction` package that make up the `AuctionServlet` servlet and `AuctionItemBean` Enterprise Bean. Because they are all to be installed in an `auction` directory accessible to the production Enterprise JavaBeans server, bundle them together so they can be unpacked in one step in the destination directory and placed in the `acution` subdirectory..

```
auction.AuctionServlet.class
auction.AuctionItem.class
auction.AuctionItemBean.class
auction.AuctionItemHome.class
auction.AuctionItemPK.class
auction.DeploymentDescriptor.txt
AuctionItemBeanHomeImpl_ServiceStub.class
WLStub1h1153e3h2r4x3t5w6e82e6jd412c.class
WLStub364c363d622h2j1j422a4oo2gm5o.class
WLSkel1h1153e3h2r4x3t5w6e82e6jd412c.class
WLSkel364c363d622h2j1j422a4oo2gm5o.class
```

Here is how to bundle them. Everything goes on one line, and the command is executed one directory above where the class files are located.

Unix:

```
jar cvf auction.jar auction/*.class
```

Win32:

```
jar cvf auction.jar auction\*.class
```

Once the JAR file is copied to the destination directory for the Enterprise beans, unpack it as follows. The extraction creates an `auction` directory with the class files in it.

```
jar xv auction.jar
```

registration Package

Here are the application files in the `registration` package that make up the `Registration` Enterprise Bean.

```
registration.Registration.class
registration.RegistrationBean.class
registration.RegistrationHome.class
registration.RegistrationPK.class
auction.DeploymentDescriptor.txt
RegistrationBeanHomeImpl_ServiceStub.class
WLStub183w4u1f4e70p6j1r4k6z1x3f6yc21.class
WLStub4z67s6n4k3sx131y4fi6w4x616p28.class
WLSkel183w4u1f4e70p6j1r4k6z1x3f6yc21.class
WLSkel4z67s6n4k3sx131y4fi6w4x616p28.class
```

Here is how to bundle them. Everything goes on one line, and the command is executed one directory above where the class files are located.

Unix:

```
jar cvf registration.jar registration/*.class
```

Win32:

```
jar cvf registration.jar registration\*.class
```

Once the JAR file is copied to the destination directory for the Enterprise beans, unpack it as follows. The extraction creates a `registration` directory with the class files in it.

```
jar xv registration.jar
```

bidder Package

Here are the application files in the `bidder` package that make up the `Bidder` Enterprise Bean.

```
bidder.Bidder.class
bidder.BidderHome.class
bidder.BidderBean.class
auction.DeploymentDescriptor.txt
BidderBeanEOImpl_ServiceStub.class
BidderBeanHomeImpl_ServiceStub.class
```

WLStub1z35502726376oa1m4m395m4w5j1j5t.class

WLStub5g4v1dm3m271tr4i5s4b4k6p376d5x.class

WLSkel1z35502726376oa1m4m395m4w5j1j5t.class

WLSkel5g4v1dm3m271tr4i5s4b4k6p376d5x.class

Here is how to bundle them. Everything goes on one line, and the command is executed one directory above where the class files are located.

Unix:

```
jar cvf bidder.jar bidder/*.class
```

Win32:

```
jar cvf bidder.jar bidder\*.class
```

Once the JAR file is copied to the destination directory for the Enterprise beans, unpack it as follows. The extraction creates a `bidder` directory with the class files in it.

```
jar xv bidder.jar
```

seller Package

Here are the application files in the `seller` package that make up the `Seller` Enterprise Bean.

`seller.Seller.class`

`seller.SellerHome.class`

`seller.SellerBean.class`

`auction.DeploymentDescriptor.txt`

`SellerBeanEOImpl_ServiceStub.class`

`SellerBeanHomeImpl_ServiceStub.class`

WLStub3xr4e731e6d2x3b3w5b693833v304q.class

WLStub86w3x4p2x6m4b696q4kjp4p4p3b33.class

WLSkel3xr4e731e6d2x3b3w5b693833v304q.class

WLSkel86w3x4p2x6m4b696q4kjp4p4p3b33.class

Here is how to bundle them. Everything goes on one line, and the command is executed one directory above where the class files are located.

Unix:

```
jar cvf seller.jar seller/*.class
```

Win32:

```
jar cvf seller.jar seller\*.class
```

Once the JAR file is copied to the destination directory for the Enterprise beans, unpack it as follows. The extraction creates a `seller` directory with the class files in it.

```
jar xv seller.jar
```

Bundle and Deploy the Administration Applet

The Administration applet family of files consists of the [AdminApplet.java](#) and [polfile.java](#) files.

Here is the `jar` command to bundle them. Everything goes on one line, and the command is executed where the policy file is located which is one directory above where the class files are located.

Unix:

```
jar cvf applet.jar admin/*.class polfile.java
```

Win32:

```
jar cvf applet.jar admin\*.class polfile.java
```

To deploy the applet, copy the `applet.jar` file to the destination applet directory and extract it as follows. The extraction creates an `admin` directory with the Administration applet class files in it.

```
jar xf applet.jar
```

Note: The applet uses JDK 1.2 APIs. It needs a policy file to access the printer and Java Plug-In to run in a pre-JDK 1.2 browser. Information on running the applet with Java Plug-In and a security policy file can be found in the [Solaris Platform](#) and [Win32 Platform](#) sections that follow.

[\[TOP\]](#)

Printable Page 

[This page was updated: 4-Jun-2001]

[Products & APIs](#) | [Developer Connection](#) | [Docs & Training](#) | [Online Support](#)
[Community Discussion](#) | [Industry News](#) | [Solutions Marketplace](#) | [Case Studies](#)

[Glossary](#) | [Feedback](#) | [A-Z Index](#)

For more information on Java technology and other software from Sun Microsystems, call: (800) 786-7638
Outside the U.S. and Canada, dial your country's [AT&T Direct Access Number](#) first.



Copyright © 1995-2001 [Sun Microsystems, Inc.](#)
All Rights Reserved. [Terms of Use](#). [Privacy Policy](#).



- Products & APIs
- Developer Connection
- Docs & Training
- Online Support
- Community Discussion
- Industry News
- Solutions Marketplace
- Case Studies

[Training Index](#)

Writing Advanced Applications

Chapter 9 continued: Solaris™ Operating System

[<<BACK] [CONTENTS] [NEXT>>]

Printable Page 

Java™ Plug-in software lets you direct applets or JavaBeans™ components on intranet web pages to run using the Java Runtime Environment (JRE) instead of the web browser's default virtual machine. The Java Plug-In works with Netscape Communicator and Microsoft Internet Explorer.

■ Requires login

Free downloads of all the software you need to install and use Java Plug-In are available from the [download](#) page.

Early Access ■
Downloads

Bug Database ■
Submit a Bug
View Database

Newsletters
Back Issues
Subscribe

Learning Centers
Articles
Bookshelf
Code Samples
New to Java

Question of the Week
Quizzes
Tech Tips
Tutorials

Forums

[Get Downloads](#)

[Extract Downloaded Files](#)

[Install Java Plug-In](#)

[Install Java Plug-In Patches](#)

[Install Netscape Communicator](#)

[Check the Installation](#)

[Install the HTML Converter](#)

[Security Policy Files](#)

- [Types of Policy Files](#)
- [Installing the Policy File](#)
- [Changing the Name or Location](#)
- [Run the Administration Applet](#)

Get Downloads

To install and use Java Plug-In on Solaris™ 2.6 or Solaris 7, you need the following downloads. Put the downloads in a directory anywhere you want.

Java Plug-In for Solaris operating systems. It is available for either Intel or Sparc platforms.

Technology Centers

Java Plug-In patches for either Solaris 2.6 or Solaris 7, depending on which one you have.

Netscape Communicator 4.5.1 (webstart version).

Java Plug-In HTML Converter

These instructions were tested on a Sun Microsystems Ultra 2 running Solaris 2.6 with Netscape Communicator 4.5.1.

Extract Downloaded Files

Go to the directory where you downloaded the files and extract each one.

Extract Java Plug-In Files:

```
zcat plugin-12-webstart-sparc.tar.Z | tar -xf -
```

Extract Patch Files for Solaris 2.6:

```
zcat JPI1.2-Patches-Solaris2.6-sparc.tar.Z | tar -xf -
```

Extract Netscape Navigator 4.5.1:

```
zcat NSCPcom_webstart_sparc.tar.Z | tar -xf -
```

Install Java Plug-In

The Java Plug-In download includes a user guide that you can view in your browser from the following directory:

```
plugin-12-webstart-sparc/Java_Plug-in_1.2.2/  
common/Docs/en/Users_Guide_Java_Plug-in.html
```

The user guide explains how to install Java Plug-In. There are several easy ways to do it, and the command sequence below is one quick way that installs Java Plug-In in the default `/opt/NSCPcom` directory using the `pkgadd` command:

```
su  
<root password>  
cd ~/plugin-12-webstart-sparc  
pkgadd -d ./Java_Plug-in_1.2.2/sparc/Product
```

Install Java Plug-In Patches

Before you can run Java Plug-In, you have to install the patches. You install the patches one at a time as root. The following command sequence goes to the patch directory, lists the files, and issues the command to install the first patch:

```
cd ~/JPI1.2-Patches-Solaris2.6-sparc  
su  
<password>  
kq6py#ls  
105210-19 105490-07 105568-13
```

```
kq6py#./105210-19/installpatch 105210-19
```

You will see this output when the patch is successfully installed:

```
Patch number 105210-19 has beenZ successfully
installed.
See /var/sadm/patch/105210-19/log for details
```

```
Patch packages installed:
```

```
SUNWarc
SUNWcsu
```

Continue installing the patches one-by-one until all patches have successfully installed. The user's guide provides a list of required and suggested patches and links to where you can download additional suggested patches if you want to install them.

Install Netscape Communicator

The extracted Netscape Communicator 4.5.1 files provide a user's guide in the `/home/monicap/NETSCAPE/Netscape_Communicator_4.51/common/Docs/en` directory that explains the installation. The following command sequence is one easy way to do it with the `pkgadd` command. By default, the installation puts Netscape Communicator in the `/opt/NSCPcom` directory where your Java Plug-In and patches are also installed.

When you extracted the `NSCPcom_webstart_sparc.tar.Z` download, it placed the files in a `NETSCAPE` directory. From the `NETSCAPE` directory, execute the following command sequence:

```
cd ~/NETSCAPE/Netscape_Communicator_4.51/sparc/Product
su
<password>
pkgadd -d .
```

Check the Installation

There are two ways to check your Java Plug-In, patch, and Netscape Communicator installation.

- 1. Open the Netscape Help menu and select About Plug_Ins. You will see a list of Mime types. Check this list against the list presented in the user's guide. If your mime types are correct, the installation is correct and complete.**
- 2. Start the control panel applet by loading the `/opt/NSCPcom/j2pi/ControlPanel.html` file. If the applet starts, the installation is correct and complete.**

The control panel lets you change the default settings used by Java Plug-In at startup. All applets running inside Java Plug-In use these settings.


```
cd /opt/NSCPcom/j2pi  
ControlPanel &
```

Install the HTML Converter

Your browser will not automatically use the Java Plug-In when you load an HTML file with an applet. You have to download and run the Java Plug-In HTML Converter on the HTML page that invokes the applet to direct the applet to run using the plug-in instead of the browser's default runtime.

Unzip the Java Plug-In HTML Converter download:

```
unzip htmlconv12.zip
```

Add the `HTMLConverter.java` program or its directory to your CLASSPATH.

Security Policy File

The auction application uses an applet running in a browser for administrative operations. In the Java™ 2 platform, applets are restricted to a sandbox-like environment and need permission to access system resources outside their restricted environment. Applets are restricted to read operations within their local directory. All other access operations require permission.

Types of Policy Files

You need a policy file to grant access permissions to the Administration applet. If the applet runs on a disk other than the disk where the browser is running, the applet will also need to be signed. See [Signed Applets](#) for information on signing and deploying applets.

There are three kinds of policy files: system, user, and program. The system policy file is located in `jdk1.2/jre/lib/security/java.policy` or `jre1.2/lib/security/java.policy` and contains permissions for everyone on the system.

The user policy file is located in the user's home directory. The user policy file provides a way to give certain users additional permissions over those granted to everyone on the system. The permissions in the system file are combined with the permissions in the user file.

A program policy file can be located anywhere. It is specifically

named when an application is invoked with the `java` command or when an applet is invoked with `applet viewer`. When an application or applet is invoked with a specific policy file, the permissions in that policy file take the place of (are not combined with) permissions specified in the system or user policy file. Program policy files are used for program testing or intranet deployment of applets and applications.

Installing the Policy File

Place the security policy file in your home directory and name it `.java.policy`. When the applet tries to perform an action that requires a policy file with a permission, the policy file is loaded from this directory and remains in effect until you exit and restart the browser.

If an applet tries to perform an access operation without the right permission, it quietly quits without raising either an applet or a browser error.

Changing the Name or Location

You can change the name and/or location of the default system or user policy file. Edit the `jdk1.2/jre/lib/security/java.security` or `jre1.2/lib/security/java.security` file and add a third entry specifying the name and location of an alternative policy file.

```
policy.url.1=
  file:${java.home}/lib/security/java.policy
policy.url.2=file:${user.home}/.java.policy
policy.url.3=file:/<mypolicyfile path and name>
```

Run the Administration Applet

Copy the Java Archive (JAR) file with the Administration applet and policy file to its final location. In this example, that location is the `/home/zelda/public_html` directory. Next, extract the applet class file and policy file from the JAR file:

```
cp admin.jar /home/zelda/public_html
jar xf applet.jar
```

The extraction places the policy file under `public_html` and creates an `admin` directory under the `public_html` directory with the applet class file in it. Rename the policy file in the `public_html` directory to `.java.policy` and copy it to your home directory.

In the `public_html` directory, create an HTML file that invokes the Administration applet class. Be sure to include the `admin` directory

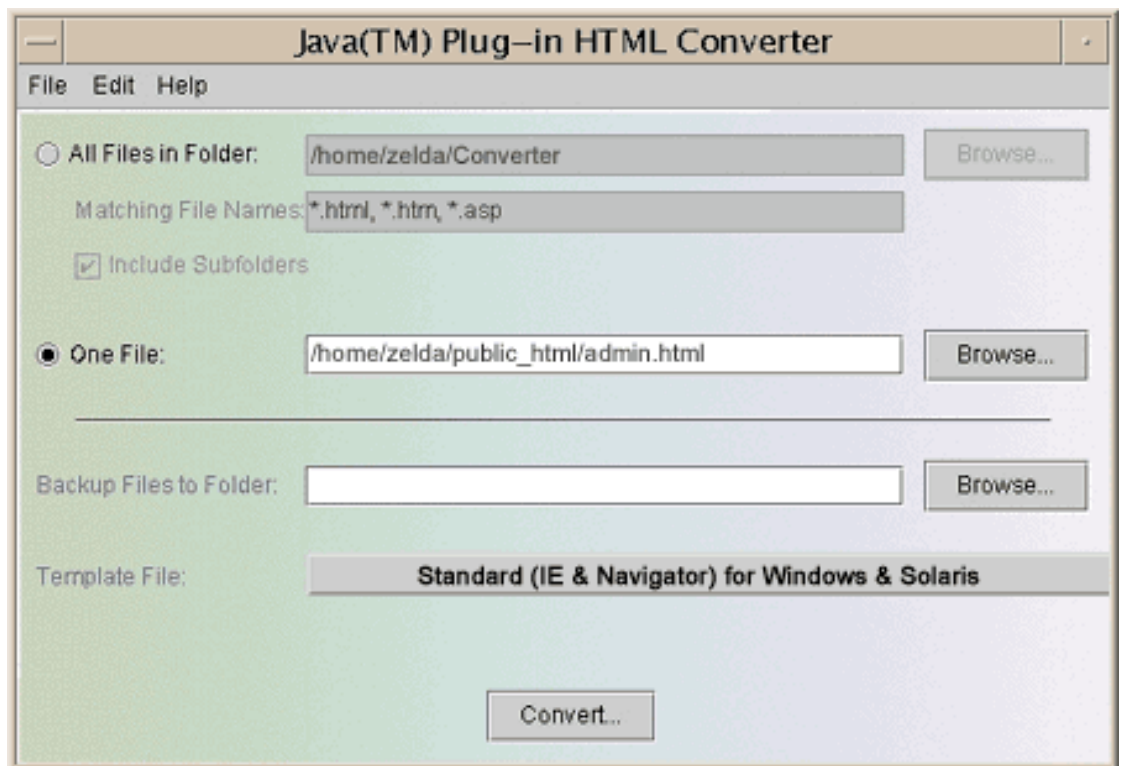
when you specify the applet class to the `CODE` option. Note that when using Java Plug-In, you cannot have the browser load the class file from the Java Archive (JAR) file.

```
<HTML>
<BODY>
<APPLET CODE=admin/AdminApplet.class
  WIDTH=550
  HEIGHT=150>
</APPLET>
</BODY>
</HTML>
```

Start the HTML Converter.

```
java HTMLConverter
```

In the HTML Converter graphical user interface, select `One File:`, specify the path to the `admin.html` file, and click the `Convert` button.



After the conversion completes, load the `admin.html` file in your browser.

[\[TOP\]](#)

Printable Page 

[This page was updated: 4-Jun-2001]

[Products & APIs](#) | [Developer Connection](#) | [Docs & Training](#) | [Online Support](#)
[Community Discussion](#) | [Industry News](#) | [Solutions Marketplace](#) | [Case Studies](#)

[Glossary](#) | [Feedback](#) | [A-Z Index](#)

For more information on Java technology
and other software from Sun Microsystems, call:
(800) 786-7638
Outside the U.S. and Canada, dial your country's
[AT&T Direct Access Number](#) first.



Copyright © 1995-2001 [Sun Microsystems, Inc.](#)
All Rights Reserved. [Terms of Use.](#) [Privacy Policy.](#)



- [Products & APIs](#)
- [Developer Connection](#)
- [Docs & Training](#)
- [Online Support](#)
- [Community Discussion](#)
- [Industry News](#)
- [Solutions Marketplace](#)
- [Case Studies](#)

[Training Index](#)

Writing Advanced Applications

Chapter 9 continued: Win32 Platform

[\[<<BACK\]](#) [\[CONTENTS\]](#) [\[NEXT>>\]](#)

[Printable Page](#) 

■ Requires login

[Early Access](#) ■

[Downloads](#)

[Bug Database](#) ■

[Submit a Bug](#)

[View Database](#)

[Newsletters](#)

[Back Issues](#)

[Subscribe](#)

[Learning Centers](#)

[Articles](#)

[Bookshelf](#)

[Code Samples](#)

[New to Java](#)

[Question of the Week](#)

[Quizzes](#)

[Tech Tips](#)

[Tutorials](#)

[Forums](#)

On Win32 platforms, Java™ Plug-In software is bundled with the Java 2 Runtime Environment. Java Plug-In lets web browsers use the Java 2 Runtime Environment to run 1.2-based applets and JavaBeans™ components instead of the web browser's default virtual machine. The Java Plug-In works with Netscape Communicator and Microsoft Internet Explorer.

[Get Downloads](#)

[Install JRE with Java Plug-In](#)

[Install HTML Converter](#)

[Install the Security Policy File](#)

- [Types of Policy Files](#)
- [Installing the Policy File](#)
- [Changing the Name or Location](#)

[Run the Administration Applet](#)

[How Does it Work?](#)

Get Downloads

To install and use the Java Runtime Environment with Java Plug-In, you need the following downloads. Put the downloads in a temporary directory.

Java Runtime Environment with Java Plug-In for Win32 Platforms.

Java Plug-In HTML Converter

Install JRE with Java Plug-In

An optionally installable version of the Java 2 Runtime Environment with Java Plug-In is included with the [Java 2 SDK](#)

Technology Centers download. You can also download and install Java 2 Runtime Environment with Java Plug-In [separately](#).

Either way, install the Java 2 Runtime Environment with Java Plug-In by double-clicking its icon and following the installation instructions. When the installation completes, you will see the Java Plug-In control panel on your Windows *Start* menu under *Programs*.

Install the HTML Converter

Your browser will not automatically use the Java Plug-In when you load an HTML file with an applet. You have to download and run the Java Plug-In HTML Converter on the HTML page that invokes the applet to direct the applet to run using the plug-in instead of the browser's default runtime.

Unzip the Java Plug-In HTML Converter download:

```
unzip htmlconv12.zip
```

Add the `HTMLConverter.java` program or its directory to your `CLASSPATH`.

Security Policy File

The auction application uses an applet running in a browser for administrative operations. In the Java™ 2 platform, applets are restricted to a sandbox-like environment and need permission to access system resources outside their restricted environment. Applets are restricted to read operations within their local directory. All other access operations require permission.

Types of Policy Files

You need a policy file to grant access permissions to the Administration applet. If the applet runs on a disk other than the disk where the browser is running, the applet will also need to be signed. See [Signed Applets](#) for information on signing and deploying applets.

There are three kinds of policy files: system, user, and program. The system policy file is located in `jdk1.2\jre\lib\security\java.policy` or `jre1.2\lib\security/java.policy` and contains permissions for everyone on the system.

The user policy file is located in the user's home directory. The user policy file provides a way to give certain users additional

permissions over those granted to everyone on the system. The permissions in the system file are combined with the permissions in the user file.

A program policy file can be located anywhere. It is specifically named when an application is invoked with the `java` command or when an applet is invoked with `applet viewer`. When an application or applet is invoked with a specific policy file, the permissions in that policy file take the place of (are not combined with) permissions specified in the system or user policy file. Program policy files are used for program testing or intranet deployment of applets and applications.

Install the Security Policy File

Place the security policy file in your home directory and name it `java.policy`. When the applet tries to perform an action that requires a policy file with a permission, the policy file is loaded from this directory and remains in effect until you exit and restart the browser.

If an applet tries to perform an access operation without the right permission, it quietly quits without raising either an applet or a browser error.

Changing the Name or Location

You can change the name and/or location of the default system or user policy file. Edit the `jdk1.2\jre\lib\security\java.security` or `jre1.2\lib\security\java.security` file and add a third entry specifying the name and location of an alternative policy file.

```
policy.url.1=file:${java.home}\lib\security\java.policy
policy.url.2=file:${user.home}\java.policy
policy.url.3=file:\<mypolicyfile path and name>
```

Note: On Windows/NT machines, you might place the policy file in the `C:\Winnt\Profiles\<userid>\java.policy` directory.

Run the Administration Applet

Copy the Java Archive (JAR) file with the Administration applet and policy file to its final location. In this example, that location is the `\home\zelda\public_html` directory. Next, extract the applet class file and policy file from the JAR file:

```
cp admin.jar \home\zelda\public_html
jar xf applet.jar
```

The extraction places the policy file under `public_html` and creates an `admin` directory under the `public_html` directory with the applet class file in it. Rename the policy file in the `public_html` directory to `java.policy` and copy it to your home directory.

In the `public_html` directory, create an HTML file that invokes the Administration applet class. Be sure to include the `admin` directory when you specify the applet class to the `CODE` option. Note that when using Java Plug-In, you cannot have the browser load the class file from the Java Archive (JAR) file.

```
<HTML>
<BODY>
<APPLET CODE=admin/AdminApplet.class
  WIDTH=550
  HEIGHT=150>
</APPLET>
</BODY>
</HTML>
```

Start the HTML Converter.

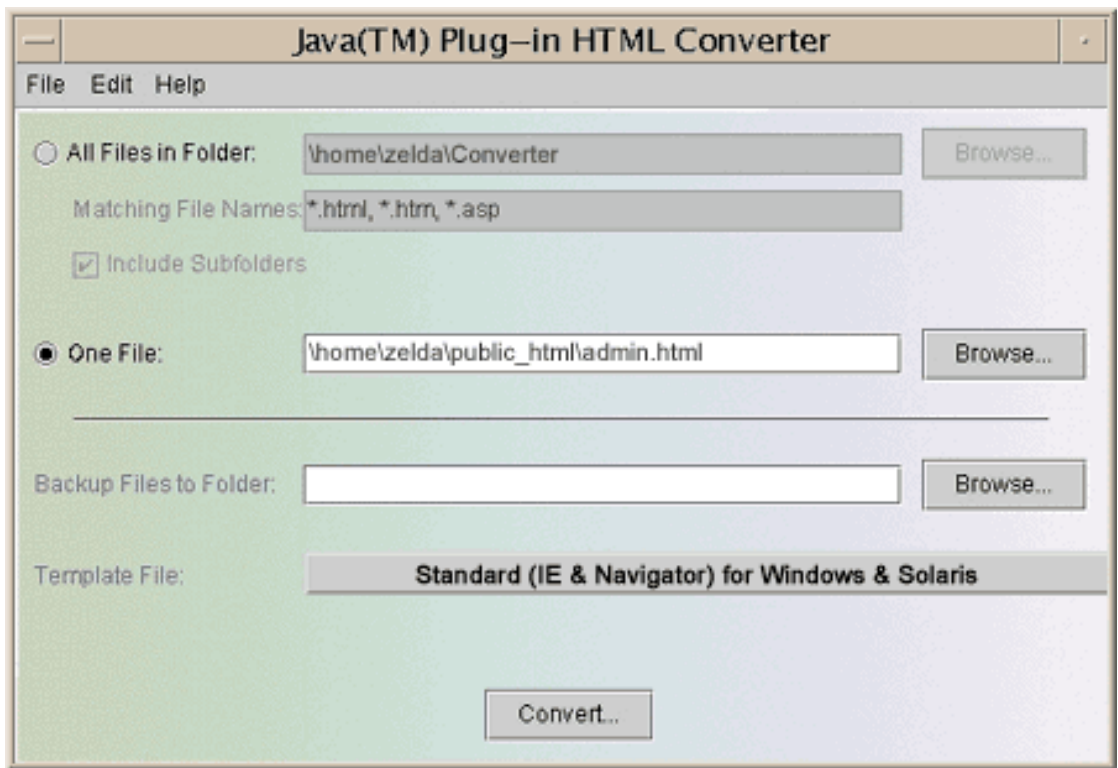
```
java HTMLConverter
```

In the HTML Converter graphical user interface, select `One File:`, specify the path to the `admin.html` file, and click the `Convert` button.

How Does It Work?


On Windows machines, the Java Plug-In finds the Java Runtime Environment (JRE) by running the OLE custom control file `beans.ocx` installed by default in the `\Program Files\JavaSoft\1.2\bin` web browser directory. The OLE control examines the Windows registry to find the Java Plug-In key and uses the value associated with that key to find the installed JRE.

If you find that the wrong JRE is being loaded, use `regedit` to check the Java Plug-In registry values for the current user. If no JRE is installed, the control checks the Java Plug-in values for `HKEY_LOCAL_MACHINE`. You should see a value for Java Runtime Environment under `Software\JavaSoft`.



After the conversion completes, load the `admin.html` file in your browser.

[\[TOP\]](#)

[Printable Page](#) 

[This page was updated: 4-Jun-2001]

[Products & APIs](#) | [Developer Connection](#) | [Docs & Training](#) | [Online Support](#)
[Community Discussion](#) | [Industry News](#) | [Solutions Marketplace](#) | [Case Studies](#)

[Glossary](#) | [Feedback](#) | [A-Z Index](#)

For more information on Java technology
and other software from Sun Microsystems, call:
(800) 786-7638
Outside the U.S. and Canada, dial your country's
[AT&T Direct Access Number](#) first.



Copyright © 1995-2001 [Sun Microsystems, Inc.](#)
All Rights Reserved. [Terms of Use](#). [Privacy Policy](#).



- Products & APIs
- Developer Connection
- Docs & Training
- Online Support
- Community Discussion
- Industry News
- Solutions Marketplace
- Case Studies

[Training Index](#)

Writing Advanced Applications

Chapter 10: More Security Topics

[\[<<BACK\]](#) [\[CONTENTS\]](#) [\[NEXT>>\]](#)

This chapter presents two additional security topics that you might find of interest.

- [Signed Applets](#)
- [Writing a Security Manager](#)

[Printable Page](#)

■ Requires login

[Early Access](#) ■
[Downloads](#)

[Bug Database](#) ■
[Submit a Bug](#)
[View Database](#)

[Newsletters](#)
[Back Issues](#)
[Subscribe](#)

[Learning Centers](#)
[Articles](#)
[Bookshelf](#)
[Code Samples](#)
[New to Java](#)
[Question of the Week](#)
[Quizzes](#)
[Tech Tips](#)
[Tutorials](#)

[Forums](#)


In a Rush?

This table links you directly to specific topics.

Topic	Section
Signed Applets	Signed Applet Example Intranet Developer End User Running an Application with a Policy File Signed Applets in JDK 1.1
Writing a Security Manager	The FileIO Program The PasswordSecurityManager Program Run The FileIO Program Reference Information

[\[TOP\]](#)

Technology Centers

Printable Page 

[This page was updated: 4-Jun-2001]

[Products & APIs](#) | [Developer Connection](#) | [Docs & Training](#) | [Online Support](#)
[Community Discussion](#) | [Industry News](#) | [Solutions Marketplace](#) | [Case Studies](#)

[Glossary](#) | [Feedback](#) | [A-Z Index](#)

For more information on Java technology
and other software from Sun Microsystems, call:
(800) 786-7638
Outside the U.S. and Canada, dial your country's
[AT&T Direct Access Number](#) first.



Copyright © 1995-2001 [Sun Microsystems, Inc.](#)
All Rights Reserved. [Terms of Use.](#) [Privacy Policy.](#)



- Products & APIs
- Developer Connection
- Docs & Training
- Online Support
- Community Discussion
- Industry News
- Solutions Marketplace
- Case Studies

[Training Index](#)

Writing Advanced Applications

Chapter 10: Signed Applets

[<<BACK] [CONTENTS] [NEXT>>]

Printable Page 

■ Requires login

Early Access ■
[Downloads](#)

Bug Database ■
[Submit a Bug](#)
[View Database](#)

Newsletters
[Back Issues](#)
[Subscribe](#)

Learning Centers
[Articles](#)
[Bookshelf](#)
[Code Samples](#)
[New to Java](#)
[Question of the Week](#)
[Quizzes](#)
[Tech Tips](#)
[Tutorials](#)

[Forums](#)

A policy file can be defined to require a signature on all applets or applications that attempt to run with the policy file. The signature is a way to verify that the applet or application is from a reliable source and can be trusted to run with the permissions granted in the policy file.

If a policy file requires a signature, an applet or application can get the access granted by the policy file only if it has the correct signature. If the applet or application has the wrong signature or no signature, it will not get access to the file.

This section walks through an example of signing an applet, verifying the signature, and running the applet with a policy file.

[Signed Applet Example](#)

[Intranet Developer](#)

[End User](#)

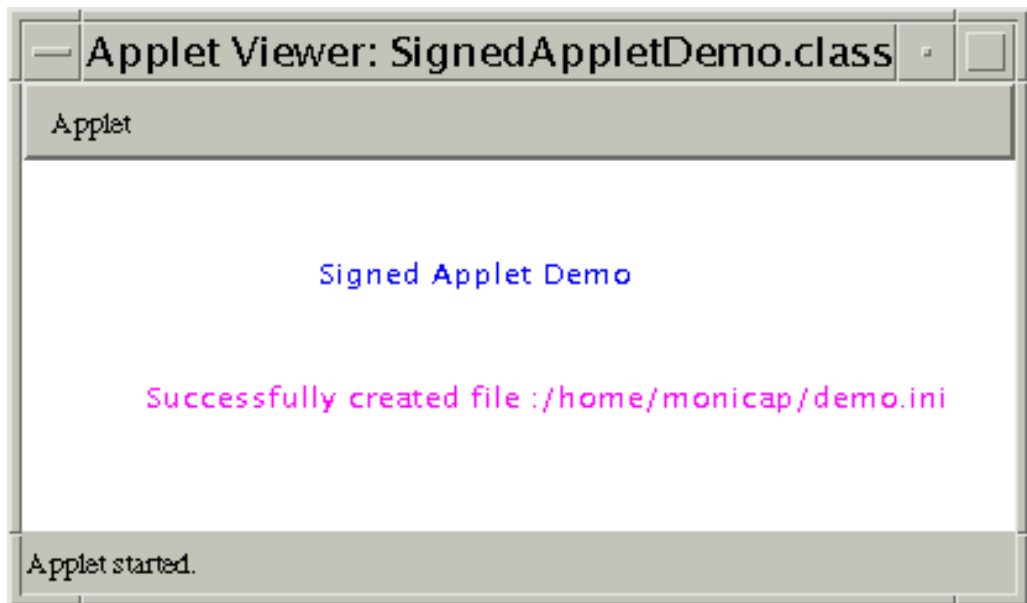
[Running an Application with a Policy File](#)

[Signed Applets in JDK 1.1](#)

Signed Applet Example

The policy file granting access can be set up to require or not require a signature. If a signature is required, the applet has to be bundled into a Java ARchive (JAR) file before it can be signed. This example shows you how to sign and grant permission to an applet so it can create `demo.ini` in the user's home directory when it executes in Applet Viewer.

Technology Centers



These files are used for the example. You can copy them to or create them in your working directory.

[SignedAppletDemo.java](#) file containing the applet code
[Write.jp](#) policy file granting access to the user's home directory

Applet tag embedded in the SignedApplet.html file:

```
<applet code="SignedAppletDemo.class"
        archive="SSignedApplet.jar"
        width=400 height=400>
    <param name=file value="/etc/inet/hosts">
</applet>
```

Usually an applet is bundled and signed by an intranet developer and handed off to the end user who verifies the signature and runs the applet. In this example, the intranet developer performs Steps 1 through 5 and Ray, the end user, performs Steps 6 through 8. But, to keep things simple, all steps occur in the same working directory.

1. Compile the applet
2. Create a JAR file
3. Generate Keys
4. Sign the JAR file
5. Export the Public Key Certificate
6. Import the Certificate as a Trusted Certificate
7. Create the policy file
8. Run the applet

Intranet Developer

Susan, the intranet developer, bundles the applet executable in a JAR file, signs the JAR file, and exports the public key certificate.

1: Compile the Applet

In her working directory, Susan uses the `javac` command to compile the `SignedAppletDemo.java` class. The output from the `javac` command is the `SignedAppletDemo.class`.

```
javac SignedAppletDemo.java
```

2: Make a JAR File

Susan then stores the compiled `SignedAppletDemo.class` file into a JAR file. The `-cvf` option to the `jar` command creates a new archive (c), using verbose mode (v), and specifies the archive file name (f). The archive file name is `SignedApplet.jar`.

```
jar cvf SignedApplet.jar SignedAppletDemo.class
```

3: Generate Keys

A JAR file is signed with the private key of the creator of the JAR file and the signature is verified by the recipient of the JAR file with the public key in the pair. The certificate is a statement from the owner of the private key that the public key in the pair has a particular value so the person using the public key can be assured the public key is authentic. Public and private keys must already exist in the keystore database before `jarsigner` can be used to sign or verify the signature on a JAR file.

Susan creates a keystore database named `compstore` that has an entry for a newly generated public and private key pair with the public key in a certificate using the `keytool` command.

In her working directory, Susan creates a keystore database and generates the keys:

```
keytool -genkey -alias signFiles -keystore compstore  
-keypass kpi135 -dname "cn=jones"  
-storepass ab987c
```

This `keytool -genkey` command invocation generates a key pair that is identified by the alias `signFiles`. Subsequent `keytool` command invocations use this alias and the key password (`-keypass kpi135`) to access the private key in the generated pair.

The generated key pair is stored in a keystore database called `compstore` (`-keystore compstore`) in the current directory, and accessed with the `compstore` password (`-storepass ab987c`).

The `-dname "cn=jones"` option specifies an X.500 Distinguished Name with a `commonName (cn)` value. X.500 Distinguished Names identify entities for X.509 certificates. In this example, Susan uses her last name, Jones, for the common name. She could use any common name that suits her purposes.

You can view all keytool options and parameters by typing:

```
keytool -help
```

4: Sign the JAR File

JAR Signer is a command line tool for signing and verifying the signature on JAR files. In her working directory, Susan uses `jarsigner` to make a signed copy of the `SignedApplet.jar` file.

```
jarsigner -keystore compstore -storepass ab987c  
          -keypass kpi135  
          -signedjar  
          SSignedApplet.jar SignedApplet.jar signFiles
```

The `-storepass ab987c` and `-keystore compstore` options specify the keystore database and password where the private key for signing the JAR file is stored. The `-keypass kpi135` option is the password to the private key, `SSignedApplet.jar` is the name of the signed JAR file, and `signFiles` is the alias to the private key. `jarsigner` extracts the certificate from the keystore whose entry is `signFiles` and attaches it to the generated signature of the signed JAR file.

5: Export the Public Key Certificate

The public key certificate is sent with the JAR file to the end user who will be using the applet. That person uses the certificate to authenticate the signature on the JAR file. A certificate is sent by exporting it from the `compstore` database.

In her working directory, Susan uses `keytool` to copy the certificate from `compstore` to a file named `CompanyCer.cer` as follows:

```
keytool -export -keystore compstore -storepass ab987c  
        -alias signFiles -file CompanyCer.cer
```

As the last step, Susan posts the JAR and certificate files to a distribution directory on a web page.

End User

Ray, the end user, downloads the JAR file from the distribution

directory, imports the certificate, creates a policy file granting the applet access, and runs the applet.

6: Import Certificate as a Trusted Certificate

Ray downloads `SSignedApplet.jar` and `CompanyCer.cer` to his home directory. Ray must now create a keystore database (`raystore`) and import the certificate into it using the alias `company`. Ray uses `keytool` in his home directory to do this:

```
keytool -import -alias company -file
        CompanyCer.cer -keystore
        raystore -storepass abcdefgh
```

7: Create the Policy File

The policy file grants the `SSignedApplet.jar` file signed by the alias `company` permission to create `demo.ini` (and no other file) in the user's home directory.

Ray creates the policy file in his home directory using either `policytool` or an ASCII editor.

```
keystore "/home/ray/raystore";

// A sample policy file that lets a program
// create demo.ini in user's home directory
// Satya N Dodda

grant SignedBy "company" {
    permission java.util.PropertyPermission
        "user.home", "read";
    permission java.io.FilePermission
        "${user.home}/demo.ini", "write";
};
```

8: Run the Applet in Applet Viewer

Applet Viewer connects to the HTML documents and resources specified in the call to `appletviewer`, and displays the applet in its own window. To run the example, Ray copies the signed JAR file and HTML file to `/home/aURL/public_html` and invokes Applet viewer from his home directory as follows:

```
appletviewer -J-Djava.security.policy=Write.jp
             http://aURL.com/SignedApplet.html
```

Note: Type everything on one line and put a space after

Write.jp

The `-J-Djava.security.policy=Write.jp` option tells Applet Viewer to run the applet referenced in the `SignedApplet.html` file with the `Write.jp` policy file.

Note: The Policy file can be stored on a server and specified in the `appletviewer` invocation as a URL.

Running an Application with a Policy File


This application invocation restricts `MyProgram` to a sandbox-like environment the same way applets are restricted, but allows access as specified in the `polfile` policy file.

```
java -Djava.security.manager
      -Djava.security.policy=polfile MyProgram
```

Signed Applets in JDK 1.1

JDK 1.1 signed applets can access local system resources if the local system is properly set up to allow it. See the JDK 1.1 [Signed Applet Example](#) page for details.

[\[TOP\]](#)

Printable Page 

[This page was updated: 4-Jun-2001]

[Products & APIs](#) | [Developer Connection](#) | [Docs & Training](#) | [Online Support](#)
[Community Discussion](#) | [Industry News](#) | [Solutions Marketplace](#) | [Case Studies](#)

[Glossary](#) | [Feedback](#) | [A-Z Index](#)

For more information on Java technology and other software from Sun Microsystems, call: (800) 786-7638
Outside the U.S. and Canada, dial your country's [AT&T Direct Access Number](#) first.



Copyright © 1995-2001 [Sun Microsystems, Inc.](#)
All Rights Reserved. [Terms of Use](#). [Privacy Policy](#).



- Products & APIs
- Developer Connection
- Docs & Training
- Online Support
- Community Discussion
- Industry News
- Solutions Marketplace
- Case Studies

[Training Index](#)

Writing Advanced Applications

Chapter 10 Continued: Writing a Security Manager

[\[<<BACK\]](#) [\[CONTENTS\]](#) [\[NEXT>>\]](#)

Printable Page 

A security manager is a Java™ virtual machine (VM) object that implements a security policy. By default, the Java 2® platform software provides a security manager that disallows all access to local system resources apart from read and write access to the directory and its subdirectories where the program is invoked.

■ Requires login

Early Access ■
[Downloads](#)

Bug Database ■
[Submit a Bug](#)
[View Database](#)

Newsletters
[Back Issues](#)
[Subscribe](#)

Learning Centers
[Articles](#)
[Bookshelf](#)
[Code Samples](#)
[New to Java](#)
[Question of the Week](#)
[Quizzes](#)
[Tech Tips](#)
[Tutorials](#)

Forums

You can extend the default security manager to implement customized verifications and approvals for applets and applications, but the implementation must include the appropriate access verification code for every `checkXXX` method you override. If you do not include this code, no access verification check happens, and your code breaches the system security policy.

This section uses an example application to explain how to write a custom security manager that prompts the end user for password identification before reading from and writing to specific files. The implementation includes access verification code so once the end user makes it through the password check, he or she still needs the file read and write permissions in their policy file.

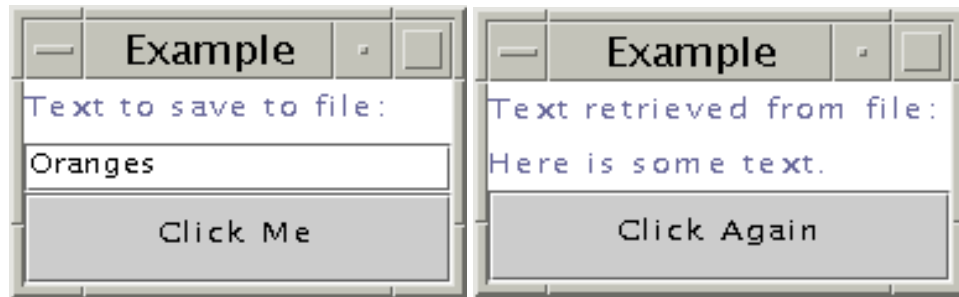
The example consists of the `FileIO` application, and the `PasswordSecurityManager` program that provides the custom security manager implementation.

- [The FileIO Program](#)
- [The PasswordSecurityManager Program](#)
- [Run The FileIO Program](#)
- [Reference Information](#)

The FileIO Program

The [FileIO](#) program displays a simple user interface and asks the

Technology Centers end user to enter some text. When the end user clicks the `Click Me` button, the text is saved to a file in the end user's home directory, and a second file is opened and read. The text read from the second file is displayed to the end user.



Before Button Click

After Button Click

The custom security manager for this program prompts the end user to enter a password before it allows `FileIO` to write text to or read text from a file. The `main` method of `FileIO` creates a custom security manager called `PasswordSecurityManager`.

```
public static void main(String[] args){
    BufferedReader buffy = new BufferedReader(
        new InputStreamReader(System.in));
    try {
        System.setSecurityManager(
            new PasswordSecurityManager("pwd", buffy));
    } catch (SecurityException se) {
        System.err.println("SecurityManager already set!");
    }
}
```

The PasswordSecurityManager Class

The [PasswordSecurityManager](#) class declares two private instance variables, which are initialized by the constructor when the custom security manager is installed. The `password` instance variable contains the actual password, and the `buffy` instance variable is an input buffer that stores the end user's password input.

```
public class PasswordSecurityManager
    extends SecurityManager{

    private String password;
    private BufferedReader buffy;

    public PasswordSecurityManager(String p,
        BufferedReader b){
        super();
        this.password = p;
        this.buffy = b;
    }
}
```

The `accessOK` method prompts the end user for a password, verifies

the password, and returns true if the password is correct and false if it is not.

```
private boolean accessOK() {
    int c;
    String response;

    System.out.println("Password, please:");
    try {
        response = buffy.readLine();
        if (response.equals(password))
            return true;
        else
            return false;
    } catch (IOException e) {
        return false;
    }
}
```

Verify Access

The SecurityManager parent class provides methods to verify file system read and write access. The checkRead and checkWrite methods each have a version that accepts a String and another version that accepts a file descriptor.

This example overrides only the String versions to keep the example simple, and because the FileIO program accesses directories and files as Strings.

```
public void checkRead(String filename) {
    if((filename.equals(File.separatorChar + "home" +
        File.separatorChar + "monicap" +
        File.separatorChar + "text2.txt"))){
        if(!accessOK()){
            super.checkRead(filename);
            throw new SecurityException("No Way!");
        } else {
            FilePermission perm = new FilePermission(
                File.separatorChar + "home" +
                File.separatorChar + "monicap" +
                File.separatorChar + "text2.txt", "read");
            checkPermission(perm);
        }
    }
}

public void checkWrite(String filename) {
    if((filename.equals(File.separatorChar + "home" +
        File.separatorChar + "monicap" +
        File.separatorChar + "text.txt"))){
        if(!accessOK()){
            super.checkWrite(filename);
            throw new SecurityException("No Way!");
        } else {
```

```
        FilePermission perm = new FilePermission(
            File.separatorChar + "home" +
            File.separatorChar + "monicap" +
            File.separatorChar + "text.txt" ,
            "write");
        checkPermission(perm);
    }
}
}
```

The `checkWrite` method is called before the end user input is written to the output file. This is because the `FileOutputStream` class calls `SecurityManager.checkWrite` first.

The custom implementation for `SecurityManager.checkWrite` tests for the pathname `/home/monicap/text.txt`, if `true` prompts the end user for the password. If the password is correct, the `checkWrite` method performs the access check by creating an instance of the required permission and passing it to the `SecurityManager.checkPermission` method. This check will succeed if the security manager finds a system, user, or program policy file with the specified permission. Once the write operation completes, the end user is prompted for the password two more times. The first time to read the `/home/monicap` directory, and the second time to read the `text2.txt` file. An access check is performed before the read operation takes place.

Policy File

Here is the policy file the `FileIO` program needs for its read and write operations. It also grants permission to the custom security manager to access the event queue on behalf of the application and show the application window without the warning banner.

```
grant {
    permission java.io.FilePermission
        "${user.home}/text.txt", "write";
    permission java.util.PropertyPermission
        "user.home", "read";
    permission java.io.FilePermission
        "${user.home}/text2.txt", "read";
    permission java.awt.AWTPermission
        "accessEventQueue";
    permission java.awt.AWTPermission
        "showWindowWithoutWarningBanner";
};
```

Run the FileIO Program

Here is how to run the `FileIO` program with the policy file:

```
java -Djava.security.policy=polfile FileIO
```

Reference Information

[Appendix A: Security and Permissions](#) describes the available permissions and explains the consequences of granting permissions. One way to use this information is to help you limit what permissions a given applet or application might need to successfully execute. Another way to use this information is to educate yourself on the ways in which a particular permission can be exploited by malicious code.

[Appendix B: Classes, Methods, and Permissions](#) provides lists of Java 2 platform software methods that are implemented to perform security access checks, the permission each requires, and the `java.security.SecurityManager` method called to perform the access check.

You can use this reference to write your own security manager implementations or when you implement abstract methods that perform security-related tasks.

[Appendix C: SecurityManager Methods](#) lists the permissions checked for by the `SecurityManager` methods.

[\[TOP\]](#)

Printable Page 

[This page was updated: 4-Jun-2001]

[Products & APIs](#) | [Developer Connection](#) | [Docs & Training](#) | [Online Support](#)
[Community Discussion](#) | [Industry News](#) | [Solutions Marketplace](#) | [Case Studies](#)

[Glossary](#) | [Feedback](#) | [A-Z Index](#)

For more information on Java technology
and other software from Sun Microsystems, call:
(800) 786-7638
Outside the U.S. and Canada, dial your country's
[AT&T Direct Access Number](#) first.



Copyright © 1995-2001 [Sun Microsystems, Inc.](#)
All Rights Reserved. [Terms of Use](#). [Privacy Policy](#).



- Products & APIs
- Developer Connection
- Docs & Training
- Online Support
- Community Discussion
- Industry News
- Solutions Marketplace
- Case Studies

[Training Index](#)

Writing Advanced Applications

Appendix A: Security and Permissions

[<<BACK] [CONTENTS] [NEXT>>]

Printable Page 

■ Requires login

Early Access ■
[Downloads](#)

Bug Database ■
[Submit a Bug](#)
[View Database](#)

Newsletters
[Back Issues](#)
[Subscribe](#)

Learning Centers
[Articles](#)
[Bookshelf](#)
[Code Samples](#)
[New to Java](#)
[Question of the Week](#)
[Quizzes](#)
[Tech Tips](#)
[Tutorials](#)

Forums

All applets and any applications invoked with a security manager must be granted explicit permission to access local system resources apart from read access to the directory and its subdirectories where the program is invoked. The Java™ platform provides permissions to allow various levels of access to different types of local information.

Because permissions let an applet or application override the default security policy, you should be very careful when you assign permissions to not create an opening for malicious code to attack your system.

This appendix describes the available permissions and explains how each permission can create an opening for malicious attacks. One way to use this information is to help you limit what permissions a given applet or application might need to successfully execute. Another way to use this information is to educate yourself on the ways in which a particular permission can be exploited by malicious code.

As a safeguard, never trust an unknown applet or application. Always check the code carefully against the information in this appendix to be sure you are not giving malicious code permission to cause serious problems on the local system.

- [Overview](#)
- [Knowing Which Permissions](#)
- [AllPermission](#)
- [AWTPermission](#)
- [FilePermission](#)
- [NetPermission](#)
- [PropertyPermission](#)
- [ReflectPermission](#)

Overview

Permissions are granted to a program with a policy file. A policy file contains permissions for specific access. A permission consists of the permission name, a target, and in some cases, a comma-separated list of actions.

For example, the following policy file entry specifies a `java.io.FilePermission` permission that grants read access (the action) to the `${user.home}/text2.txt` target.

```
grant {  
    permission java.io.FilePermission  
        "${user.home}/text2.txt", "read";  
};
```

There is one policy file for Java platform installation (system) and an optional policy file for each user. The system policy file is in `{java.home}/lib/security/java.policy`, and the user policy file is in each user's home directory. The system and user policy files are combined. So for example, there could be a system policy file with very few permissions granted to all users on the system, and individual policy files granting additional permissions to certain users.

To run an application with the security manager and a policy file named `polfile` in the user's home directory, type:

```
java -Djava.security.manager  
-Djava.security.policy=polfile FileIO
```

To run an applet in appletviewer with a policy file named `polfile` in the user's home directory, type:

```
appletviewer  
-J-Djava.security.policy=polfile fileIO.html
```

When running an applet in a browser, the browser looks for the user and system policy files to find the permissions the applet needs to access local system resources on behalf of the user who downloaded the applet.

Knowing Which Permissions

When you run an applet or invoke an application with a security manager that needs permissions, you will get a stack trace if you did not provide a policy file with all the needed permissions. The stack trace contains the information you need to add the permission to the policy file that caused the stack trace. If the program needs additional permissions, you will keep getting stack traces until all the required permissions are added to the policy file. The only drawback to this approach is you have to try every possible code path in your application.

Another way to determine which permission your program needs is to browse [Appendix B: Methods and Permissions](#). This appendix tells you which Java 2 platform software methods are prevented from executing without the listed permission. The information in Appendix B is also useful for developers who want to write their own security manager to customize the verifications and approvals needed in a program.

Here is a short example to show you how to translate the first couple of lines in a stack trace to a policy file entry. The first line tells you access is denied. This means this stack trace was generated because the program tried to access a system resource without the proper permission. The second line means you need a `java.net.SocketPermission` that gives the program permission to connect to and resolve the host name for Internet Protocol (IP) address `129.144.176.176`, port `1521`.

```
java.security.AccessControlException: access denied
    (java.net.SocketPermission
     129.144.176.176:1521 connect,resolve)
```

To turn this into a policy file entry, list the permission name, a target, and an action list as follows where

`java.net.SocketPermission` is the permission name, `129.144.176.176:1521` is the target, and `connect,resolve` is the action list.

```
grant {
    permission java.net.SocketPermission
        "129.144.176.176:1521", "connect,resolve";
};
```

AllPermission

`java.security.AllPermission` specifies all permissions in the system

for all possible targets and actions. This permission should be used only during testing because it grants permission to run with all security restrictions disabled as if there were no security manager.

```
grant {  
    permission java.security.AllPermission;  
};
```

AWTPermission

`java.awt.AWTPermission` grants access to the following Abstract Window Toolkit (AWT) targets. The possible targets are listed by name with no action list.

```
grant {  
    permission java.awt.AWTPermission  
        "accessClipboard";  
    permission java.awt.AWTPermission  
        "accessEventQueue";  
    permission java.awt.AWTPermission  
        "showWindowWithoutWarningBanner";  
};
```

accessClipboard: This target grants permission to post information to and retrieve information from the AWT clipboard. Granting this permission could allow malicious code to share potentially sensitive or confidential information.

accessEventQueue: This target grants permission to access the AWT event queue. Granting this permission could allow malicious code to peek at and remove existing events from the system, or post bogus events that could cause the application or applet to perform malicious actions.

listenToAllAWTEvents: This target grants permission to listen to all AWT events throughout the system. Granting this permission could allow malicious code to read and exploit confidential user input such as passwords.

Each AWT event listener is called from within the context of that event queue's `EventDispatchThread`, so if the `accessEventQueue` permission is also enabled, malicious code could modify the contents of AWT event queues throughout the system, which can cause the application or applet to perform unintended and malicious actions.

readDisplayPixels: This target grants permission to read pixels back from the display screen. Granting this permission could allow interfaces such as `java.awt.Composite` that allow arbitrary code to examine pixels on the display to include malicious code that

snoops on user activities.

showWindowWithoutWarningBanner: This target grants permission to display a window without also displaying a banner warning that the window was created by an applet. Without this warning, an applet might pop up windows without the user knowing they belong to an applet. This could be a problem in environments where users make security-sensitive decisions based on whether the window belongs to an applet or an application. For example, disabling the banner warning might trick the end user into entering sensitive user name and password information.

FilePermission

`java.io.FilePermission` grants access to a file or directory. The targets consist of the target pathname and a comma-separated list of actions.

This policy file grants read, write, delete, and execute permission to all files.

```
grant {
    permission java.io.FilePermission
        "<<ALL FILES>>", "read, write, delete, execute";
};
```

This policy file grants read and write permission to `text.txt` in the user's home directory.

```
grant {
    permission java.io.FilePermission
        "${user.home}/text.txt", "read, write";
};
```

You can use the following wild cards to specify the target pathname.

A pathname that ends in `/*`, where `/` is the file separator character indicates a directory and all the files contained in that directory.

A pathname that ends with `/-` indicates a directory, and recursively, all files and subdirectories contained in that directory.

A pathname consisting of a single asterisk (`*`) indicates all files in the current directory.

A pathname consisting of a single dash (-) indicates all files in the current directory, and recursively, all files and subdirectories contained in the current directory.

The actions are specified in a list of comma-separated keywords and have the following meanings:

read: Permission to read a file or directory.

write: Permission to write to and create a file or directory.

execute: Permission to execute a file or search a directory.

delete: Permission to delete a file or directory.

When granting file permissions, always think about the implications of granting read and especially write access to various files and directories. The <<ALL FILES>> permission with write action is especially dangerous because it grants permission to write to the entire file system. This means the system binary can be replaced, which includes the Java¹ Virtual Machine (VM) runtime environment.

NetPermission

`java.net.NetPermission` grants access to various network targets. The possible targets are listed by name with no action list.

```
grant {
    permission java.net.NetPermission
        "setDefaultAuthenticator";
    permission java.net.NetPermission
        "requestPasswordAuthentication";
};
```

`setDefaultAuthenticator`: This target grants permission to set the way authentication information is retrieved when a proxy or HTTP server asks for authentication. Granting this permission could mean malicious code can set an authenticator that monitors and steals user authentication input as it retrieves the input from the user.

`requestPasswordAuthentication`: This target grants permission to ask the authenticator registered with the system for a password. Granting this permission could mean malicious code might steal the password.

`specifyStreamHandler`: This target grants permission to specify a stream handler when constructing a Uniform Resource Locator (URL). Granting this permission could mean malicious code might create a URL with resources to which it would not normally have access, or specify a stream handler that gets the actual bytes from somewhere to which it does have access. This means the malicious

code could trick the system into creating a `ProtectionDomain/CodeSource` for a class even though the class really did not come from that location.

PropertyPermission

`java.util.PropertyPermission` grants access to system properties. The `java.util.Properties` class represents persistent settings such as the location of the installation directory, the user name, or the user's home directory.

```
grant {
    permission java.util.PropertyPermission
        "java.home", "read";
    permission java.util.PropertyPermission
        "os.name", "write";
    permission java.util.PropertyPermission
        "user.name", "read, write";
};
```

The target list contains the name of the property; for example, `java.home` or `os.name`. The naming convention for the properties follows the hierarchical property naming convention, and includes wild cards. An asterisk at the end of the property name, after a dot (`.`), or alone signifies a wild card match. For example, `java.*` or `*` are valid, but `*java` or `a*b` are invalid.

The actions are specified in a list of comma-separated keywords, and have the following meanings:

read: Permission to read (get) a property.

write: Permission to write (set) a property.

Granting property permissions can leave your system open to intrusion. For example, granting permission to access the `java.home` property makes the installation directory vulnerable to attack, and granting permission to access the `user.name` and `user.home` properties might reveal the user's account name and home directory to code that might misuse the information.

ReflectPermission

`java.lang.reflect.ReflectPermission` grants permission for various reflective operations. The possible targets are listed by name with no action list.

```
grant {
    permission java.lang.reflect.ReflectPermission
        "suppressAccessChecks";
};
```

suppressAccessChecks: This target grants permission to access fields and invoke methods in a class. This includes public, protected, and private fields and methods. Granting this permission could reveal confidential information and make normally unavailable methods accessible to malicious code.

RuntimePermission

`java.lang.RuntimePermission` grants access to various runtime targets such as the class loader, Java VM, and thread. The possible targets are listed by name with no action list.

```
grant {
    permission java.lang.RuntimePermission
        "createClassLoader";
    permission java.lang.RuntimePermission
        "getClassLoader";
    permission java.lang.RuntimePermission
        "exitVM";
    permission java.lang.RuntimePermission
        "setFactory";
    permission java.lang.RuntimePermission
        "setIO";
    permission java.lang.RuntimePermission
        "modifyThread";
    permission java.lang.RuntimePermission
        "modifyThreadGroup";
    permission java.lang.RuntimePermission
        "getProtectionDomain";
    permission java.lang.RuntimePermission
        "setProtectionDomain";
    permission java.lang.RuntimePermission
        "readFileDescriptor";
    permission java.lang.RuntimePermission
        "writeFileDescriptor";
    permission java.lang.RuntimePermission
        "loadLibrary.<library name>";
    permission java.lang.RuntimePermission
        "accessClassInPackage.<package name>";
    permission java.lang.RuntimePermission
        "defineClassInPackage.<package name>";
    permission java.lang.RuntimePermission
        "accessDeclaredMembers.<class name>";
    permission java.lang.RuntimePermission
        "queuePrintJob";
};
```

The naming convention for target information where a library, package, or class name is added follows the hierarchical property naming convention, and includes wild cards. An asterisk at the end of the target name, after a dot (`.`), or alone signifies a wild card match. For example, `loadLibrary.*` or `*` are valid, but `*loadLibrary` or `a*b` are not.

createClassLoader: This target grants permission to create a class loader. Granting this permission might allow a malicious application to instantiate its own class loader and load harmful classes into the system. Once loaded, the class loader could place these classes into any protection domain and give them full permissions for that domain.

getClassLoader: This target grants permission to retrieve the class loader for the calling class. Granting this permission could enable malicious code to get the class loader for a particular class and load additional classes.

setContextClassLoader: This target grants permission to set the context class loader used by a thread. System code and extensions use the context class loader to look up resources that might not exist in the system class loader. Granting this permission allows code to change which context class loader is used for a particular thread, including system threads. This can cause problems if the context class loader has malicious code.

setSecurityManager: This target grants permission to set or replace the security manager. The security manager is a class that allows applications to implement a security policy. Granting this permission could enable malicious code to install a less restrictive manager, and thereby, bypass checks that would have been enforced by the original security manager.

createSecurityManager: This target grants permission to create a new security manager. Granting this permission could give malicious code access to protected and sensitive methods that might disclose information about other classes or the execution stack. It could also allow the introduction of a weakened security manager.

exitVM: This target grants permission to halt the Java VM. Granting this permission could allow malicious code to mount a denial-of-service attack by automatically forcing the VM to stop.

setFactory: This target grants permission to set the socket factory used by the `ServerSocket` or `Socket` class, or the stream handler factory used by the `URL` class. Granting this permission allows code to set the actual implementation for the socket, server socket, stream handler, or Remote Method Invocation (RMI) socket factory. An attacker might set a faulty implementation that mangles the data stream.

setIO: This target grants permission to change the value of the

`System.out`, `System.in`, and `System.err` standard system streams. Granting this permission could allow an attacker to change `System.in` to steal user input, or set `System.err` to a null output stream, which would hide any error messages sent to `System.err`.

`modifyThread`: This target grants permission to modify threads by calls to the `stop`, `suspend`, `resume`, `setPriority`, and `setName` methods in the `Thread` class. Granting this permission could allow an attacker to start or suspend any thread in the system.

`stopThread`: This target grants permission to stop threads. Granting this permission allows code to stop any thread in the system provided the code already has permission to access that thread. Malicious code could corrupt the system by killing existing threads.

`modifyThreadGroup`: This target grants permission to modify threads by calls to the `destroy`, `resume`, `setDaemon`, `setMaxPriority`, `stop`, and `suspend` methods of the `ThreadGroup` class. Granting this permission could allow an attacker to create thread groups and set their run priority.

`getProtectionDomain` This target grants permission to retrieve the `ProtectionDomain` instance for a class. Granting this permission allows code to obtain policy information for that code source. While obtaining policy information does not compromise the security of the system, it does give attackers additional information, such as local file names for example, to better aim an attack.

`readFileDescriptor`: This target grants permission to read file descriptors. Granting this permission allows code to read the particular file associated with the file descriptor, which is dangerous if the file contains confidential data.

`writeFileDescriptor`: This target grants permission to write file descriptors. Granting this permission allows code to write to the file associated with the descriptor, which is dangerous if the file descriptor points to a local file.

`loadLibrary.{ library name }` : This target grants permission to dynamically link the specified library. Granting this permission could be dangerous because the security architecture is not designed to and does not extend to native code loaded by way of the `java.lang.System.loadLibrary` method.

`accessClassInPackage.{ package name }` This target grants permission to access the specified package by way of a class

loader's `loadClass` method when that class loader calls the `SecurityManager.checkPackageAccess` method. Granting this permission gives code access to classes in packages to which it normally does not have access. Malicious code may use these classes to help in its attempt to compromise security in the system.

`defineClassInPackage.{ package name}` : This target grants permission to define classes in the specified package by way of a class loader's `defineClass` method when that class loader calls the `SecurityManager.checkPackageDefinition` method. Granting this permission allows code to define a class in a particular package, which can be dangerous because malicious code with this permission might define rogue classes in trusted packages like `java.security` or `java.lang`, for example.

`accessDeclaredMembers`: This target grants permission to access the declared members of a class. Granting this permission allows code to query a class for its public, protected, default (package), and private fields and methods. Although the code would have access to the private and protected field and method names, it would not have access to the private and protected field data and would not be able to invoke any private methods. Nevertheless, malicious code may use this information to better aim an attack. Additionally, malicious code might invoke any public methods or access public fields in the class, which could be dangerous if the code would normally not be able to invoke those methods or access the fields because it cannot cast the object to the class or interface with those methods and fields.

`queuePrintJob`: This target grants permission to initiate a print job request. Granting this permission could allow code to print sensitive information to a printer or maliciously waste paper.

SecurityPermission

`java.security.SecurityPermission` grants access to various security configuration parameters. The possible targets are listed by name with no action list. Security permissions currently apply to methods called on the following objects:

`java.security.Policy`, which represents the system security policy for applications.

`java.security.Security`, which centralizes all security properties and common security methods. It manages providers.

`java.security.Provider`, which represents an implementation of such things as security algorithms (DSA, RSA, MD5, or SHA-1) and key generation.

`java.security.Signer`, which manages private keys. Even though, `Signer` is deprecated, the related permissions are available for backwards compatibility.

`java.security.Identity`, which manages real-world objects such as people, companies, or organizations whose identities can be authenticated using their public keys.

```
grant {
    permission java.security.SecurityPermission
        "getPolicy";
    permission java.security.SecurityPermission
        "setPolicy";
    permission java.security.SecurityPermission
        "getProperty.os.name";
    permission java.security.SecurityPermission
        "setProperty.os.name";
    permission java.security.SecurityPermission
        "insertProvider.SUN";
    permission java.security.SecurityPermission
        "removeProvider.SUN";
    permission java.security.SecurityPermission
        "setSystemScope";
    permission java.security.SecurityPermission
        "setIdentityPublicKey";
    permission java.security.SecurityPermission
        "setIdentityInfo";
    permission java.security.SecurityPermission
        "addIdentityCertificate";
    permission java.security.SecurityPermission
        "removeIdentityCertificate";
    permission java.security.SecurityPermission
        "clearProviderProperties.SUN";
    permission java.security.SecurityPermission
        "putProviderProperty.<provider name>";
    permission java.security.SecurityPermission
        "removeProviderProperty.SUN";
    permission java.security.SecurityPermission
        "getSignerPrivateKey";
    permission java.security.SecurityPermission
        "setSignerKeyPair";
};
```

getPolicy: This target grants permission to retrieve the system-wide security policy. Granting this permission discloses which permissions would be granted to a given application or applet. While revealing the policy does not compromise the security of the system, it does provide malicious code with additional information it could use to better aim an attack.

setPolicy: This target grants permission to set the system-wide security policy. Granting this permission could allow malicious code to grant itself all the necessary permissions to successfully mount an attack on the system.

getProperty.{ key} : This target grants permission to retrieve the security property specified by {key}. Depending on the particular key for which access has been granted, the code may have access to the list of security providers and the location of the system-wide and user security policies. While revealing this information does not compromise the security of the system, it does provide malicious code with additional information which it may use to better aim an attack.

setProperty.{ key} : This target grants permission to set the security property specified by {key}. This could include setting a security provider or defining the location of the the system-wide security policy. Malicious code that has permission to set a new security provider may set a rogue provider that steals confidential information such as cryptographic private keys. In addition, malicious code with permission to set the location of the system-wide security policy may point it to a security policy that grants the attacker all the necessary permissions it requires to successfully mount an attack on the system.

insertProvider.{ provider name} : This target grants permission to add the new security provider specified by {provider name}. Granting this permission allows the introduction of a possibly malicious provider that could do such things as disclose the private keys passed to it. This is possible because the `Security` object, which manages the installed providers, does not currently check the integrity or authenticity of a provider before attaching it.

removeProvider.{ provider name} : This target grants permission to remove the provider specified by {provider name}. Granting this permission could change the behavior or disable execution of other parts of the program. If a provider requested by the program has been removed, execution might fail.

setSystemScope: This target grants permission to set the system identity scope. Granting this permission could allow an attacker to configure the system identity scope with certificates that should not be trusted. This could grant code signed with those certificates privileges that would be denied by the original identity scope.

setIdentityPublicKey: This target grants permission to set the public key for an `Identity` object. If the identity is marked *trusted*,

this allows an attacker to introduce its own public key that is not trusted by the system's identity scope. This could grant code signed with that public key privileges that would be otherwise denied.

SetIdentityInfo: This target grants permission to set a general information string for an `Identity` object. Granting this permission allows attackers to set the general description for an identity. Doing so could trick applications into using a different identity than intended or prevent applications from finding a particular identity.

addIdentityCertificate: This target grants permission to add a certificate for an `Identity` object. Granting this permission allows attackers to set a certificate for an identity's public key making the public key trusted to a wider wider audience than originally intended.

removeIdentityCertificate: This target grants permission to remove a certificate for an `Identity` object. Granting this permission allows attackers to remove a certificate for an identity's public key. This could be dangerous because public key suddenly becomes considered less trustworthy than it otherwise would be.

printIdentity: This target grants permission to print out the name of a principal, the scope in which the principal is used, and whether the principal is considered *trusted* in that scope. The printed scope could be a filename, in which case it might convey local system information. For example, here is a sample printout of an identity named *carol*, who is marked not trusted in the user's identity database:

```
carol[/home/luehe/identitydb.obj][not trusted].
```

clearProviderProperties.{ provider name} This target grants permission to clear a `Provider` object so it no longer contains the properties used to look up services implemented by the provider. Granting this permission disables the lookup of services implemented by the provider. This could change the behavior or disable execution of other parts of the program that would normally utilize the `Provider`, as described under the `removeProvider.{provider name}` permission above.

putProviderProperty.{ provider name} : This target grants permission to set properties for the specified provider. The provider properties each specify the name and location of a particular service implemented by the provider. Granting this permission allows code to replace the service specification with another one with a different implementation and could be

dangerous if the new implementation has malicious code.

`removeProviderProperty.{ provider name}` : This target grants permission to remove properties from the specified provider. Granting this permission disables the lookup of services implemented by the provider making them inaccessible. Granting this permission to malicious code could allow the malicious code to change the behavior or disable execution of other parts of the program that would normally utilize the `Provider` object, as described under the `removeProvider.{provider name}` permission above.

`getSignerPrivateKey`: This target grants permission to retrieve the private key of a `Signer` object. Private keys should always be kept secret. Granting this permission could allow malicious code to use the private key to sign files and claim the signature came from the `Signer` object.

`setSignerKeyPair`: This target grants permission to set the public and private key pair for a `Signer` object. Granting this permission could allow an attacker to replace the target's key pair with a possibly weaker (smaller) key pair. This would also allow an attacker to listen in on encrypted communication between the target and its peers. The target's peers might wrap an encryption session key under the target's *new* public key, which would allow the attacker (who possesses the corresponding private key) to unwrap the session key and decipher the communication data encrypted under that session key.

SerializablePermission

`java.io.SerializablePermission` grants access to serialization operations. The possible targets are listed by name with no action list.

```
grant {
    permission java.io.SerializablePermission
        "enableSubclassImplementation";
    permission java.io.SerializablePermission
        "enableSubstitution";
};
```

`enableSubclassImplementation`: This target grants permission to implement a subclass of `ObjectOutputStream` or `ObjectInputStream` to override the default serialization or deserialization of objects. Granting this permission could allow code to use this to serialize or deserialize classes in a malicious way. For example, during serialization, malicious code could store confidential private field

data in a way easily accessible to attackers; or, during deserialization malicious code could deserialize a class with all its private fields zeroed out.

enableSubstitution: This target grants permission to substitute one object for another during serialization or deserialization. Granting this permission could allow malicious code to replace the actual object with one that has incorrect or malignant data.

SocketPermission

The `java.net.SocketPermission` permission grants access to a network by way of sockets. The target is a host name and port address, and the action list specifies ways to connect to that host. Possible connections are `accept`, `connect`, `listen`, and `resolve`.

This policy file entry allows a connection to and accepts connections on port 7777 on the host `puffin.eng.sun.com`.

```
grant {
    permission java.net.SocketPermission
        "puffin.eng.sun.com:7777",
        "connect, accept";
};
```

This policy file entry allows connections to, accepts connections on, and listens on any port between 1024 and 65535 on the local host.

```
grant {
    permission java.net.SocketPermission
        "localhost:1024-",
        "accept, connect, listen";
};
```

The host is expressed with the following syntax as a DNS name, as a numerical IP address, or as `localhost` (for the local machine). The asterisk (*) wild card can be included once in a DNS name host specification. If included, it must be in the left-most position, as in `*.sun.com`.

```
host = (hostname | IPaddress)[:portrange]
portrange = portnumber | -portnumber |
            portnumber-[portnumber]
```

The port or port range is optional. A port specification of the form `N-`, where `N` is a port number, means all ports numbered `N` and above, while a specification of the form `-N` indicates all ports numbered `N` and below.

The `listen` action is only meaningful when used with `localhost`, and the `resolve` (resolve host/ip name service lookups) action is implied when any of the other actions are present.

Granting code permission to accept or make connections to remote hosts may be dangerous because malevolent code can more easily transfer and share confidential data among parties that might not otherwise have access to the data.

Note: On Unix platforms, only root is normally allowed access to ports lower than 1024.

[\[TOP\]](#)

¹ As used on this web site, the terms "Java virtual machine" or "JVM" mean a virtual machine for the Java platform.

Printable Page 

[This page was updated: 4-Jun-2001]

[Products & APIs](#) | [Developer Connection](#) | [Docs & Training](#) | [Online Support](#)
[Community Discussion](#) | [Industry News](#) | [Solutions Marketplace](#) | [Case Studies](#)

[Glossary](#) | [Feedback](#) | [A-Z Index](#)

For more information on Java technology
and other software from Sun Microsystems, call:
(800) 786-7638
Outside the U.S. and Canada, dial your country's
[AT&T Direct Access Number](#) first.



Copyright © 1995-2001 [Sun Microsystems, Inc.](#)
All Rights Reserved. [Terms of Use](#). [Privacy Policy](#).



[Products & APIs](#)
[Developer Connection](#)
[Docs & Training](#)
[Online Support](#)
[Community Discussion](#)
[Industry News](#)
[Solutions Marketplace](#)
[Case Studies](#)

[Training Index](#)

Writing Advanced Applications

Appendix B: Classes, Methods, and Permissions

[\[<<BACK\]](#)
[\[CONTENTS\]](#)
[\[NEXT>>\]](#)

[Printable Page](#)

■ Requires login

[Early Access](#) ■
[Downloads](#)

[Bug Database](#) ■
[Submit a Bug](#)
[View Database](#)

[Newsletters](#)
[Back Issues](#)
[Subscribe](#)

[Learning Centers](#)
[Articles](#)
[Bookshelf](#)
[Code Samples](#)
[New to Java](#)
[Question of the Week](#)
[Quizzes](#)
[Tech Tips](#)
[Tutorials](#)

[Forums](#)

A number of Java™ 2 platform methods are implemented to verify access permissions. This means that before they execute, they verify that there is a [system, user, or program](#) has a policy file with the required permissions for execution to continue. If no such permission is found, execution stops with an error condition.

The access verification code passes the required permissions to the [security manager](#), and the security manager checks that permission against the policy file permissions to determine whether to access. This means that Java 2 platform API methods are associated with specific permissions, and specific permissions are associated with specific `java.security.SecurityManager` methods.

This appendix lists the Java 2 platform methods, the permission associated with each method, and the `java.security.SecurityManager` method called to verify the existence of that permission. You need this information when you implement certain abstract methods or create your own [security manager](#) so you can include access verification code to keep your implementations in line with Java 2 platform security policy. If you do not include access verification code, your implementations will bypass the built-in Java 2 platform security checks.

[java.awt.Graphics2D](#)

[java.awt.Toolkit](#)

[java.awt.Window](#)

[java.beans.Beans](#)

[java.beans.Introspector](#)

[java.beans.PropertyEditorManager](#)

[java.io.File](#)

[java.io.FileOutputStream](#)

[java.io.ObjectInputStream](#)

Technology Centers

[java.io.ObjectOutputStream](#)
[java.io.RandomAccessFile](#)
[java.lang.Class](#)
[java.lang.ClassLoader](#)
[java.lang.Runtime](#)
[java.lang.SecurityManager](#)
[java.lang.System](#)
[java.lang.Thread](#)
[java.lang.ThreadGroup](#)
[java.lang.reflect.AccessibleObject](#)
[java.net.Authenticator](#)
[java.net.DatagramSocket](#)
[java.net.HttpURLConnection](#)
[java.net.InetAddress](#)
[java.net.MulticastSocket](#)
[java.net.ServerSocket](#)
[java.net.Socket](#)
[java.net.URL](#)
[java.net.URLConnection](#)
[java.net.URLClassLoader](#)
[java.rmi.activation.ActivationGroup](#)
[java.rmi.server.RMISocketFactory](#)
[java.security.Identity](#)
[java.security.IdentityScope](#)
[java.security.Permission](#)
[java.security.Policy](#)
[java.security.Provider](#)
[java.security.SecureClassLoader](#)
[java.security.Security](#)
[java.security.Signer](#)
[java.util.Locale](#)
[java.util.Zip](#)

java.awt.Graphics2D

```

public abstract void setComposite(Composite comp)
java.Security.SecurityManager.checkPermission
java.awt.AWTPermission "readDisplayPixels"

```

The access verification code for `setComposite` should call `java.Security.SecurityManager.checkPermission` and pass it `java.awt.AWTPermission "readDisplayPixels"` when a `Graphics2D`

context draws to a Component on the display screen and the Composite is a custom object rather than an AlphaComposite object.

java.awt.Toolkit

```
public void addAWTEventListener(
    AWTEventListener listener,
    long eventMask)
public void removeAWTEventListener(
    AWTEventListener listener)
checkPermission
java.awt.AWTPermission "listenToAllAWTEvents"
```

~~~~~

```
public abstract PrintJob getPrintJob(
    Frame frame, String jobtitle,
    Properties props)
checkPrintJobAccess
java.lang.RuntimePermission "queuePrintJob"
```

~~~~~

```
public abstract Clipboard
    getSystemClipboard()
checkSystemClipboardAccess
java.awt.AWTPermission "accessClipboard"
```

~~~~~

```
public final EventQueue
    getSystemEventQueue()
checkAwtEventQueueAccess
java.awt.AWTPermission "accessEventQueue"
```

## java.awt.Window

```
Window()
checkTopLevelWindow
java.awt.AWTPermission
    "showWindowWithoutWarningBanner"
```

## java.beans.Beans

```
public static void setDesignTime(
    boolean isDesignTime)
public static void setGuiAvailable(
    boolean isGuiAvailable)
checkPropertiesAccess
java.util.PropertyPermissions "*", "read,write"
```

## java.beans.Introspector

```

public static synchronized void
    setBeanInfoSearchPath(String path[])
checkPropertiesAccess
java.util.PropertyPermissions "*", "read,write"

```

## java.beans.PropertyEditorManager

```

public static void registerEditor(
    Class targetType,
    Class editorClass)
public static synchronized void
    setEditorSearchPath(String path[])
checkPropertiesAccess
java.util.PropertyPermissions "*", "read,write"

```

## java.io.File

```

public boolean delete()
public void deleteOnExit()
checkDelete(String)
java.io.FilePermission "{name}", "delete"

```

~~~~~

```

public boolean exists()
public boolean canRead()
public boolean isFile()
public boolean isDirectory()
public boolean isHidden()
public long lastModified()
public long length()
public String[] list()
public String[] list(FilenameFilter filter)
public File[] listFiles()
public File[] listFiles(FilenameFilter filter)
public File[] listFiles(FileFilter filter)
checkRead(String)
java.io.FilePermission "{name}", "read"

```

~~~~~

```

public boolean canWrite()
public boolean createNewFile()
public static File createTempFile(
    String prefix, String suffix)
public static File createTempFile(
    String prefix, String suffix,
    File directory)
public boolean mkdir()
public boolean mkdirs()
public boolean renameTo(File dest)
public boolean setLastModified(long time)
public boolean setReadOnly()
checkWrite(String)
java.io.FilePermission "{name}", "write"

```

## java.io.FileInputStream

```
FileInputStream(FileDescriptor fdObj)
checkRead(FileDescriptor)
java.lang.RuntimePermission "readFileDescriptor"
```

~~~~~

```
FileInputStream(String name)
FileInputStream(File file)
checkRead(String)
java.io.FilePermission "{name}", "read"
```

java.io.FileOutputStream

```
FileOutputStream(FileDescriptor fdObj)
checkWrite(FileDescriptor)
java.lang.RuntimePermission "writeFileDescriptor"
```

~~~~~

```
FileOutputStream(File file)
FileOutputStream(String name)
FileOutputStream(String name, boolean append)
checkWrite(String)
java.io.FilePermission "{name}", "write"
```

## java.io.ObjectInputStream

```
protected final boolean
    enableResolveObject(boolean enable);
checkPermission
java.io.SerializablePermission
    "enableSubstitution"
```

~~~~~

```
protected ObjectInputStream()
protected ObjectOutputStream()
checkPermission
java.io.SerializablePermission
    "enableSubclassImplementation"
```

java.io.ObjectOutputStream

```
protected final boolean
    enableReplaceObject(boolean enable)
checkPermission
java.io.SerializablePermission
    "enableSubstitution"
```

java.io.RandomAccessFile

```
RandomAccessFile(String name, String mode)
RandomAccessFile(File file, String mode)
checkRead(String)
java.io.FilePermission "{name}", "read"
```

In both these methods the mode is *r*.

~~~~~

```
RandomAccessFile(String name, String mode)
checkRead(String) and checkWrite(String)
java.io.FilePermission "{name}", "read,write"
```

**In this method the mode is *rw*.**

~~~~~

java.lang.Class

```
public static Class forName(
    String name, boolean initialize,
    ClassLoader loader)
checkPermission
java.lang.RuntimePermission("getClassLoader")
```

The access verification code for this method calls `checkPermission` and pass it `java.lang.RuntimePermission("getClassLoader")` when loader is null and the caller's class loader is not null.

~~~~~

```
public Class[] getClasses()
checkMemberAccess(this, Member.DECLARED)
java.lang.RuntimePermission
    "accessDeclaredMembers"
java.lang.RuntimePermission
    "accessClassInPackage.{pkgName}"
```

**The access verification code for this class and each of its superclasses calls `checkMemberAccess(this, Member.DECLARED)`. If the class is in a package, `checkPackageAccess({pkgName})` is also called. By default, `checkMemberAccess` does not require permission if this class's classloader is the same as that of the caller. Otherwise, it requires `java.lang.RuntimePermission "accessDeclaredMembers"`. If the class is in a package, `java.lang.RuntimePermission "accessClassInPackage.{pkgName}"` is also required.**

~~~~~

```
public ClassLoader getClassLoader()
checkPermission
java.lang.RuntimePermission "getClassLoader"
```

If the caller's class loader is null, or is the same as or an ancestor of the class loader for the class whose class loader is being requested, no permission is needed. Otherwise, `java.lang.RuntimePermission "getClassLoader"` is required.

~~~~~

```
public Class[] getDeclaredClasses()
public Field[] getDeclaredFields()
public Method[] getDeclaredMethods()
public Constructor[]
    getDeclaredConstructors()
public Field getDeclaredField(
    String name)
public Method getDeclaredMethod(...)
public Constructor
    getDeclaredConstructor(...)
checkMemberAccess(this, Member.DECLARED)
checkPackageAccess({pkgName})
java.lang.RuntimePermission
    "accessDeclaredMembers"
java.lang.RuntimePermission
    "accessClassInPackage.{pkgName}"
```

**If Class is in a package, the access verification code should call `checkPackageAccess({pkgName})` and pass it**

`java.lang.RuntimePermission "accessClassInPackage.{pkgName}"`.

**If Class is not in a package, the access verification code for these methods should call `checkMemberAccess(this, Member.DECLARED)` and pass it**

`java.lang.RuntimePermission "accessClassInPackage.{pkgName}"`.

~~~~~

```
public Field[] getFields()
public Method[] getMethods()
public Constructor[] getConstructors()
public Field getField(String name)
public Method getMethod(...)
public Constructor getConstructor(...)
checkMemberAccess(this, Member.PUBLIC)
checkPackageAccess({pkgName})
java.lang.RuntimePermission
    "accessClassInPackage.{pkgName}"
```

If Class is not in a package, the access verification code for these methods calls `checkMemberAccess(this, Member.PUBLIC)`, but no permission is passed.

If Class is in a package, the access verification code for these methods should call `checkPackageAccess({pkgName})` and pass it `checkPackageAccess({pkgName})`.

~~~~~

```

public ProtectionDomain
    getProtectionDomain()
checkPermission
java.lang.RuntimePermission "getProtectionDomain"

```

## java.lang.ClassLoader

```

ClassLoader()
ClassLoader(ClassLoader parent)
checkCreateClassLoader
java.lang.RuntimePermission "createClassLoader"

```

~~~~~

```

public static ClassLoader
    getSystemClassLoader()
public ClassLoader getParent()
checkPermission
java.lang.RuntimePermission "getClassLoader"

```

If the caller's class loader is null or is the same as or an ancestor of the class loader for the class whose class loader is being requested, no permission is needed. Otherwise,

java.lang.RuntimePermission "getClassLoader" is required.

java.lang.Runtime

```

public Process exec(String command)
public Process exec(String command,
    String envp[])
public Process exec(String cmdarray[])
public Process exec(String cmdarray[],
    String envp[])
checkExec
java.io.FilePermission "{command}", "execute"

```

~~~~~

```

public void exit(int status)
public static void
    runFinalizersOnExit(boolean value)
checkExit(status) where status is 0 for
    runFinalizersOnExit
java.lang.RuntimePermission "exitVM"

```

~~~~~

```

public void load(String lib)
public void loadLibrary(String lib)
checkLink({libName})
java.lang.RuntimePermission
    "loadLibrary.{libName}"

```

In these methods {libName} is the lib, filename or libname argument.

java.lang.SecurityManager

<all methods>
 checkPermission
 See [Security Manager Methods](#).

java.lang.System

```
public static void exit(int status)
public static void
    runFinalizersOnExit(boolean value)
checkExit(status) where status is 0 for
    runFinalizersOnExit
java.lang.RuntimePermission "exitVM"
```

~~~~~

```
public static void load(String filename)
public static void loadLibrary(
    String libname)
checkLink({libName})
java.lang.RuntimePermission
    "loadLibrary.{libName}"
```

**In these methods {libName} is the lib, filename or libname argument.**

~~~~~

```
public static Properties getProperties()
public static void setProperties(Properties props)
checkPropertiesAccess
java.util.PropertyPermission "*", "read,write"
```

~~~~~

```
public static String getProperty(String key)
public static String getProperty(String key,
    String def)
checkPropertyAccess
java.util.PropertyPermission "{key}", "read"
```

~~~~~

```
public static void setIn(InputStream in)
public static void setOut(PrintStream out)
public static void setErr(PrintStream err)
checkPermission
java.lang.RuntimePermission "setIO"
```

~~~~~

```
public static String setProperty(String key,
    String value)
checkPermission
java.util.PropertyPermission "{key}", "write"
```

~~~~~

```
public static synchronized void
```



```

        setSecurityManager(SecurityManager s)
    checkPermission
    java.lang.RuntimePermission "setSecurityManager"

```

java.lang.Thread

```

public ClassLoader getContextClassLoader()
    checkPermission
    java.lang.RuntimePermission "getClassLoader"

```

If the caller's class loader is null or is the same as or an ancestor of the context class loader for the thread whose context class loader is being requested, no permission is needed. Otherwise, java.lang.RuntimePermission "getClassLoader" is required.

```

~~~~~

```

```

public void setContextClassLoader
    (ClassLoader cl)
    checkPermission
    java.lang.RuntimePermission
        "setContextClassLoader"

```

```

~~~~~

```

```

public final void checkAccess()
public void interrupt()
public final void suspend()
public final void resume()
public final void setPriority
    (int newPriority)
public final void setName(String name)
public final void setDaemon(boolean on)
    checkAccess(this)
    java.lang.RuntimePermission "modifyThread"

```

```

~~~~~

```

```

public static int
    enumerate(Thread tarray[])
    checkAccess({threadGroup})
    java.lang.RuntimePermission "modifyThreadGroup"

```

```

~~~~~

```

```

public final void stop()
    checkAccess(this).
    checkPermission
    java.lang.RuntimePermission "modifyThread"
    java.lang.RuntimePermission "stopThread"

```

The access verification code should call checkAccess and pass it java.lang.RuntimePermission "modifyThread", unless the current thread is trying to stop a thread other than itself. In this case, the access verification code should call checkPermission and pass it java.lang.RuntimePermission "stopThread".

```

~~~~~

```

```

public final synchronized void
    stop(Throwable obj)
checkAccess(this).
checkPermission
java.lang.RuntimePermission "modifyThread"
java.lang.RuntimePermission "stopThread"

```

The access verification code should call `checkAccess` and pass it `java.lang.RuntimePermission "modifyThread"` unless the current thread is trying to stop a thread other than itself or `obj` is not an instance of `ThreadDeath`. In this case, the access verification code should call `checkPermission` and pass it `java.lang.RuntimePermission "stopThread"`.

~~~~~

```

Thread()
Thread(Runnable target)
Thread(String name)
Thread(Runnable target, String name)
checkAccess({parentThreadGroup})
java.lang.RuntimePermission "modifyThreadGroup"

```

~~~~~

```

Thread(ThreadGroup group, ...)
checkAccess(this) for ThreadGroup methods, or
checkAccess(group) for Thread methods
java.lang.RuntimePermission "modifyThreadGroup"

```

java.lang.ThreadGroup

```

public final void checkAccess()
public int enumerate(Thread list[])
public int enumerate(Thread list[],
    boolean recurse)
public int enumerate(ThreadGroup list[])
public int enumerate(ThreadGroup list[],
    boolean recurse)
public final ThreadGroup getParent()
public final void
    setDaemon(boolean daemon)
public final void setMaxPriority(int pri)
public final void suspend()
public final void resume()
public final void destroy()
checkAccess(this) for ThreadGroup methods, or
checkAccess(group) for Thread methods
java.lang.RuntimePermission "modifyThreadGroup"

```

~~~~~

```

ThreadGroup(String name)
ThreadGroup(ThreadGroup parent,
String name)
checkAccess({parentThreadGroup})
java.lang.RuntimePermission "modifyThreadGroup"

```

~~~~~

```

public final void interrupt()
checkAccess(this)
java.lang.RuntimePermission "modifyThreadGroup"
java.lang.RuntimePermission "modifyThread"

```

The access verification code for this method also requires

java.lang.RuntimePermission "modifyThread" because the java.lang.Thread interrupt() method is called for each thread in the thread group and in all of its subgroups.

~~~~~

```

public final void stop()
checkAccess(this)
java.lang.RuntimePermission "modifyThreadGroup"
java.lang.RuntimePermission "modifyThread"
java.lang.RuntimePermission "stopThread"

```

**The access verification code for this method also requires**

**java.lang.RuntimePermission "modifyThread" and possibly java.lang.RuntimePermission "stopThread" because the java.lang.Thread stop() method is called for each thread in the thread group and in all of its subgroups.**

## java.lang.reflect.AccessibleObject

```

public static void setAccessible(...)
public void setAccessible(...)
checkPermission
java.lang.reflect.ReflectPermission
    "suppressAccessChecks"

```

## java.net.Authenticator

```

public static PasswordAuthentication
    requestPasswordAuthentication(InetAddress addr,
                                int port,
                                String protocol,
                                String prompt,
                                String scheme)
checkPermission
java.net.NetPermission
    "requestPasswordAuthentication"

```

~~~~~

```

public static void
    setDefault(Authenticator a)
checkPermission
java.net.NetPermission "setDefaultAuthenticator"

```

java.net.DatagramSocket

```

public void send(DatagramPacket p)
checkMulticast(p.getAddress())
checkConnect(p.getAddress().getHostAddress(),
             p.getPort())
java.net.SocketPermission((
             p.getAddress()).getHostAddress(),
             "accept,connect")
java.net.SocketPermission "{host}", "resolve"

```

The access verification code for send calls checkMulticast in the following case:

```

if (p.getAddress().isMulticastAddress()) {
    java.net.SocketPermission(
        (p.getAddress()).getHostAddress(),
        "accept,connect")
}

```

The access verification code for send calls checkConnect in the following case:

```

else {
    port = p.getPort();
    host = p.getAddress().getHostAddress();
    if (port == -1) java.net.SocketPermission
        "{host}", "resolve";
    else java.net.SocketPermission
        "{host}:{port}", "connect"
}

```

~~~~~

```

public InetAddress getLocalAddress()
checkConnect("{host}", -1)
java.net.SocketPermission "{host}", "resolve"
~~~~~

```

```

DatagramSocket(...)
checkListen({port})

```

**The access verification code for this method calls checkListen and passes in socket permissions as follows:**

```

if (port == 0)
 java.net.SocketPermission "localhost:1024-",
 "listen";
else
 java.net.SocketPermission "localhost:{port}",
 "listen"
~~~~~

```

```

public synchronized void receive(DatagramPacket p)
checkAccept("{host}", {port})
java.net.SocketPermission "{host}:{port}",
    "accept"

```

## java.net.HttpURLConnection

```
public static void setFollowRedirects(boolean set)
checkSetFactory
java.lang.RuntimePermission "setFactory"
```

## java.net.InetAddress

```
public String getHostName()
public static InetAddress[]
    getAllByName(String host)
public static InetAddress getLocalHost()
checkConnect({host}, -1)
java.net.SocketPermission "{host}", "resolve"
```

## java.net.MulticastSocket

```
public void joinGroup(InetAddress mcastaddr)
public void leaveGroup(InetAddress mcastaddr)
checkMulticast(InetAddress)
java.net.SocketPermission(
    mcastaddr.getHostAddress(),
    "accept,connect")
```

~~~~~

```
public synchronized void
    send(DatagramPacket p, byte ttl)
checkMulticast(p.getAddress(), ttl)
checkConnect(p.getAddress().getHostAddress(),
    p.getPort())
java.net.SocketPermission((
    p.getAddress()).getHostAddress(),
    "accept,connect")
java.net.SocketPermission "{host}", "resolve"
```

The access verification code for send calls checkMulticast in the following case:

```
if (p.getAddress().isMulticastAddress()) {
    java.net.SocketPermission(
        (p.getAddress()).getHostAddress(),
        "accept,connect")
}
```

The access verification code for this method calls checkConnect in the following case:

```
else {
    port = p.getPort();
    host = p.getAddress().getHostAddress();
    if (port == -1) java.net.SocketPermission
        "{host}", "resolve"
    else java.net.SocketPermission
        "{host}:{port}", "connect"
```

```

}
~~~~~

```

```

MulticastSocket(...)
checkListen({port})

```

The access verification code for this method calls `checkListen` in the following case:

```

if (port == 0)
    java.net.SocketPermission
        "localhost:1024-", "listen";
else
    java.net.SocketPermission
        "localhost:{port}", "listen"

```

java.net.ServerSocket

```

ServerSocket(...)
checkListen({port})

```

The access verification code for this method calls `checkListen` in the following case:

```

if (port == 0)
    java.net.SocketPermission
        "localhost:1024-", "listen";
else
    java.net.SocketPermission
        "localhost:{port}", "listen"

```

```

~~~~~

```

```

public Socket accept()
protected final void implAccept(Socket s)
checkAccept({host}, {port})
java.net.SocketPermission
    "{host}:{port}", "accept"

```

```

~~~~~

```

```

public static synchronized void
    setSocketFactory(...)
checkSetFactory
java.lang.RuntimePermission "setFactory"

```

java.net.Socket

```

public static synchronized void
    setSocketImplFactory(...)
checkSetFactory
java.lang.RuntimePermission "setFactory"

```

```

~~~~~

```

```

Socket(...)
checkConnect({host}, {port})
java.net.SocketPermission
    "{host}:{port}", "connect"

```

java.net.URL

```

public static synchronized void
    setURLStreamHandlerFactory(...)
checkSetFactory
java.lang.RuntimePermission "setFactory"

```

~~~~~

```

URL(...)
checkPermission
java.net.NetPermission "specifyStreamHandler"

```

## java.net.URLConnection

```

public static synchronized void
    setContentHandlerFactory(...)
public static void setFileNameMap(
    FileNameMap map)
checkSetFactory
java.lang.RuntimePermission "setFactory"

```

## java.net.URLClassLoader

```

URLClassLoader(...)
checkCreateClassLoader
java.lang.RuntimePermission "createClassLoader"

```

## java.rmi.activation.ActivationGroup

```

public static synchronized ActivationGroup
    createGroup(...)
public static synchronized void setSystem(
    ActivationSystem system)
checkSetFactory
java.lang.RuntimePermission "setFactory"

```

## java.rmi.server.RMI SocketFactory

```

public synchronized static void setSocketFactory(...)
checkSetFactory
java.lang.RuntimePermission "setFactory"

```

## java.security.Identity

```

public void addCertificate(...)
checkSecurityAccess("addIdentityCertificate")
java.security.SecurityPermission
    "addIdentityCertificate"

~~~~~

public void removeCertificate(...)
checkSecurityAccess("removeIdentityCertificate")
java.security.SecurityPermission
 "removeIdentityCertificate"

~~~~~

public void setInfo(String info)
checkSecurityAccess("setIdentityInfo")
java.security.SecurityPermission
    "setIdentityInfo"

~~~~~

public void setPublicKey(PublicKey key)
checkSecurityAccess("setIdentityPublicKey")
java.security.SecurityPermission
 "setIdentityPublicKey"

~~~~~

public String toString(...)
checkSecurityAccess("printIdentity")
java.security.SecurityPermission
    "printIdentity"

```

## java.security.IdentityScope

```

protected static void setSystemScope()
checkSecurityAccess("setSystemScope")
java.security.SecurityPermission
    "setSystemScope"

```

## java.security.Permission

```

public void checkGuard(Object object)
checkPermission(this)

```

**This Permission object is the permission checked.**

## java.security.Policy

```

public static Policy getPolicy()
checkPermission
java.security.SecurityPermission "getPolicy"

~~~~~

```



```

public static void setPolicy(Policy policy);
checkPermission
java.security.SecurityPermission "setPolicy"
~~~~~

```

## java.security.Provider

```

public synchronized void clear()
checkSecurityAccess("clearProviderProperties."
    +{name})
java.security.SecurityPermission
    "clearProviderProperties.{name}"

```

In this method *name* is the provider name.

~~~~~

```

public synchronized Object put(Object key,
    Object value)
checkSecurityAccess("putProviderProperty."
    +{name})
java.security.SecurityPermission
    "putProviderProperty.{name}"

```

In this method *name* is the provider name.

~~~~~

```

public synchronized Object remove(Object key)
checkSecurityAccess("removeProviderProperty."
    +{name})
java.security.SecurityPermission
    "removeProviderProperty.{name}"

```

In this method *name* is the provider name.

## java.security.SecureClassLoader

```

SecureClassLoader(...)
checkCreateClassLoader
java.lang.RuntimePermission "createClassLoader"

```

## java.security.Security

```

public static void getProperty(String key)
checkPermission
java.security.SecurityPermission "getProperty.{key}"

```

~~~~~

```

public static int addProvider(Provider provider)
public static int insertProviderAt(
    Provider provider,
    int position);

```

```

checkSecurityAccess("insertProvider."
                    +provider.getName())
java.security.SecurityPermission
    "insertProvider.{name}"
~~~~~

public static void removeProvider(String name)
checkSecurityAccess("removeProvider."+name)
java.security.SecurityPermission "removeProvider.{name}"
~~~~~

public static void setProperty( String key,
                               String datum)
checkSecurityAccess("setProperty."+key)
java.security.SecurityPermission
    "setProperty.{key}"

```

java.security.Signer

```

public PrivateKey getPrivateKey()
checkSecurityAccess("getSignerPrivateKey")
java.security.SecurityPermission
    "getSignerPrivateKey"

~~~~~

public final void setKeyPair(KeyPair pair)
checkSecurityAccess("setSignerKeypair")
java.security.SecurityPermission
    "setSignerKeypair"

```

java.util.Locale

```

public static synchronized void setDefault(
    Locale newLocale)
checkPermission
java.util.PropertyPermission
    "user.language","write"

```

java.util.zip.ZipFile

```

ZipFile(String name)
checkRead
java.io.FilePermission "{name}","read"

```

[\[TOP\]](#)

[Products & APIs](#) | [Developer Connection](#) | [Docs & Training](#) | [Online Support](#)
[Community Discussion](#) | [Industry News](#) | [Solutions Marketplace](#) | [Case Studies](#)

[Glossary](#) | [Feedback](#) | [A-Z Index](#)

For more information on Java technology
and other software from Sun Microsystems, call:
(800) 786-7638
Outside the U.S. and Canada, dial your country's
[AT&T Direct Access Number](#) first.



Copyright © 1995-2001 [Sun Microsystems, Inc.](#)
All Rights Reserved. [Terms of Use](#). [Privacy Policy](#).



[Products & APIs](#)
[Developer Connection](#)
[Docs & Training](#)
[Online Support](#)
[Community Discussion](#)
[Industry News](#)
[Solutions Marketplace](#)
[Case Studies](#)

[Training Index](#)

Writing Advanced Applications

Appendix C: Security Manager Methods

[<<BACK](#) | [CONTENTS](#) | [NEXT>>](#)

Printable Page 

■ Requires login

Early Access ■
[Downloads](#)

Bug Database ■
[Submit a Bug](#)
[View Database](#)

Newsletters
[Back Issues](#)
[Subscribe](#)

Learning Centers

[Articles](#)
[Bookshelf](#)
[Code Samples](#)
[New to Java](#)
[Question of the Week](#)
[Quizzes](#)
[Tech Tips](#)
[Tutorials](#)

Forums

This table shows which permissions are checked for by the default implementations of the `java.lang.SecurityManager` methods. Each of the `check` methods calls the `SecurityManager.checkPermission` method with the indicated permission, except for the `checkConnect` and `checkRead` methods that take a context argument. The `checkConnect` and `checkRead` methods expect the context to be an `AccessControlContext` and they call the context's `checkPermission` method with the specified permission.

```

public void checkAccept(String host, int port);
java.net.SocketPermission "{host}:{port}", "accept";

public void checkAccess(Thread g);
java.lang.RuntimePermission "modifyThread";

public void checkAccess(ThreadGroup g);
java.lang.RuntimePermission "modifyThreadGroup";

public void checkAwtEventQueueAccess();
java.awt.AWTPermission "accessEventQueue";

public void checkConnect(String host, int port);
if (port == -1)
    java.net.SocketPermission "{host}", "resolve";
else
    java.net.SocketPermission "{host}:{port}", "connect";

public void checkConnect(String host, int port,
    Object context);
if (port == -1)
    java.net.SocketPermission "{host}", "resolve";
else
    java.net.SocketPermission "{host}:{port}", "connect";

public void checkCreateClassLoader();
java.lang.RuntimePermission "createClassLoader";

public void checkDelete(String file);
java.io.FilePermission "{file}", "delete";
  
```

Technology Centers

```

public void checkExec(String cmd);
if (cmd is an absolute path)
    java.io.FilePermission "{cmd}", "execute";
else
    java.io.FilePermission "-", "execute";

public void checkExit(int status);
java.lang.RuntimePermission "exitVM";

public void checkLink(String lib);
java.lang.RuntimePermission "loadLibrary.{lib}";

public void checkListen(int port);
if (port == 0)
    java.net.SocketPermission "localhost:1024-","listen";
else
    java.net.SocketPermission "localhost:{port}","listen";

public void checkMemberAccess(Class clazz, int which);
if (which != Member.PUBLIC) {
    if (currentClassLoader() != clazz.getClassLoader()) {
        checkPermission(
            new java.lang.RuntimePermission(
                "accessDeclaredMembers"));
    }
}

public void checkMulticast(InetAddress maddr);
java.net.SocketPermission(
    maddr.getHostAddress(), "accept,connect");

public void checkMulticast(InetAddress maddr, byte ttl);
java.net.SocketPermission(
    maddr.getHostAddress(), "accept,connect");

public void checkPackageAccess(String pkg);
java.lang.RuntimePermission
    "accessClassInPackage.{pkg}";

public void checkPackageDefinition(String pkg);
java.lang.RuntimePermission
    "defineClassInPackage.{pkg}";

public void checkPrintJobAccess();
java.lang.RuntimePermission "queuePrintJob";

public void checkPropertiesAccess();
java.util.PropertyPermission "*", "read,write";

public void checkPropertyAccess(String key);
java.util.PropertyPermission "{key}", "read,write";

public void checkRead(FileDescriptor fd);
java.lang.RuntimePermission "readFileDescriptor";

public void checkRead(String file);
java.io.FilePermission "{file}", "read";

public void checkRead(String file, Object context);
java.io.FilePermission "{file}", "read";

public void checkSecurityAccess(String action);

```

```
java.security.SecurityPermission "{action}";

public void checkSetFactory();
java.lang.RuntimePermission "setFactory";

public void checkSystemClipboardAccess();
java.awt.AWTPermission "accessClipboard";

public boolean checkTopLevelWindow(Object window);
java.awt.AWTPermission "showWindowWithoutWarningBanner";

public void checkWrite(FileDescriptor fd);
java.lang.RuntimePermission "writeFileDescriptor";

public void checkWrite(String file);
java.io.FilePermission "{file}", "write";

public SecurityManager();
java.lang.RuntimePermission "createSecurityManager";
```

[\[TOP\]](#)

Printable Page 

[This page was updated: 4-Jun-2001]

[Products & APIs](#) | [Developer Connection](#) | [Docs & Training](#) | [Online Support](#)
[Community Discussion](#) | [Industry News](#) | [Solutions Marketplace](#) | [Case Studies](#)

[Glossary](#) | [Feedback](#) | [A-Z Index](#)

For more information on Java technology
and other software from Sun Microsystems, call:
(800) 786-7638
Outside the U.S. and Canada, dial your country's
[AT&T Direct Access Number](#) first.



Copyright © 1995-2001 [Sun Microsystems, Inc.](#)
All Rights Reserved. [Terms of Use](#). [Privacy Policy](#).



- Products & APIs
- Developer Connection
- Docs & Training
- Online Support
- Community Discussion
- Industry News
- Solutions Marketplace
- Case Studies

[Training Index](#)

Writing Advanced Applications

Epilogue

[\[<<BACK\]](#) [\[CONTENTS\]](#)

Printable Page 

■ Requires login

Early Access ■
Downloads

Bug Database ■
Submit a Bug
View Database

Newsletters
Back Issues
Subscribe

Learning Centers
Articles
Bookshelf
Code Samples
New to Java
Question of the Week
Quizzes
Tech Tips
Tutorials

Forums

Closing comments go here.

Note: This is a work in progress. Please let us know what you would like to see in this book by sending your comments and questions to us at the following address: jdcbook@sun.com

Related Materials

For more information on multithreading, see the JDC article [A Simple, Multithreaded Web Server](#).

About the Authors

Calvin Austin is a staff engineer at Sun Microsystems, and a cofounder of the Java Developer ConnectionSM (JDC). He has written articles for the JDC and presented twice at JavaOne. His main interests are CORBA, databases and Swing. In particular he is recognized for his expertise in debugging applications and analyzing stack traces.

Calvin holds an Honors degree in Computer Science from Leeds University in the United Kingdom.


[Monica Pawlan](#), a staff writer for the Java Developer ConnectionSM (JDC), is author of *Essentials of the Java Programming Language: A Hands-On Guide* (Addison-Wesley, 2000), and co-author of *Advanced Programming for the Java 2 Platform* (Addison-Wesley, 2000).

Guest Authors

Tony Squier is an engineer at Sun Microsystems and a cofounder of the JDC. He developed most of the servlets that *drive* the web site, including the session management and registration servlets. He also played a leading role in creating many JDC applets including `Chat`, `DiscussionGroup`, and others.

Tony has a Master's Degree in Computer Science with specialization in Software Engineering, a Bachelors of Arts in Communications, and a Minor in Computer Science -- all from California State University, Sacramento. Tony is a member of the Association for Computing Machinery and has interest in object oriented software development, user interface design, and client/server applications.

[\[TOP\]](#)

Printable Page 

[This page was updated: 4-Jun-2001]

[Products & APIs](#) | [Developer Connection](#) | [Docs & Training](#) | [Online Support](#)
[Community Discussion](#) | [Industry News](#) | [Solutions Marketplace](#) | [Case Studies](#)

[Glossary](#) | [Feedback](#) | [A-Z Index](#)

For more information on Java technology and other software from Sun Microsystems, call: (800) 786-7638
Outside the U.S. and Canada, dial your country's [AT&T Direct Access Number](#) first.



Copyright © 1995-2001 [Sun Microsystems, Inc.](#)
All Rights Reserved. [Terms of Use](#). [Privacy Policy](#).



[Products & APIs](#)
[Developer Connection](#)
[Docs & Training](#)
[Online Support](#)
[Community Discussion](#)
[Industry News](#)
[Solutions Marketplace](#)
[Case Studies](#)

Printable Page 

■ Requires login

Early Access ■
[Downloads](#)

Bug Database ■
[Submit a Bug](#)
[View Database](#)

Newsletters
[Back Issues](#)
[Subscribe](#)

Learning Centers
[Articles](#)
[Bookshelf](#)
[Code Samples](#)
[New to Java](#)
[Question of the Week](#)
[Quizzes](#)
[Tech Tips](#)
[Tutorials](#)

Forums

Tutorials & Short Courses

Java™ Series

[The Java Tutorial](#)
[The Swing Tutorial](#)

Java Developer ConnectionSM

[Java Embedded Server™ Technology](#)
(November 2000)
[JavaBeans™ 101 Tutorial, Part III](#)
(January 2001)
[JavaBeans™ 101 Tutorial, Part II](#)
(November 2000)
[JavaBeans™ 101 Tutorial, Part I](#)
(October 2000)
[Java 3D™ API Tutorial](#)
(September 2000)
[A New Era for Java Protocol Handlers](#)
(August 2000)
[Java™ Advanced Imaging Tutorial](#)
(July 2000)
[Writing Enterprise Applications for the Java™ 2 Enterprise Edition Reference Implementation](#)
(June 2000)
[Essentials of the Java™](#)

jGuru Short Courses



The JDC hosts a series of short courses by [jGuru.com](#) (formerly known as the MageLang Institute)

[Fundamentals of the JavaMail™ API](#)
(April 2001)
[Language Essentials Short Course](#)
(February 2001)
[JDBC™ 2.0 Fundamentals Short Course](#)
(December 2000)
[JavaServer Pages™ Fundamentals](#)
(September 2000)
[AWT Fundamentals](#)
(July 2000)
[Enterprise JavaBeans™ \(EJB™\) Technology Fundamentals](#)
(May 2000)
[Fundamentals of RMI](#)
(February 2000)
[Introduction to CORBA](#)
(December 1999)
[Introduction to the Collections Framework](#)
(September 1999)
[Introduction to the JavaBeans™ API](#)
[Effective Layout](#)

Technology Centers

Java JumpStart

Get a free copy of the Java JumpStart™ Edition from Sun Developer Essentials™ software program. This CD set includes the latest versions of JESE™, J2EE™, J2M2™, Forte™ for Java, Community Edition Java IDE, and much more.

[Programming Language: A Hands-On Guide, Part 2*](#)

(July 1999)

[Essentials of the Java™](#)

[Programming Language: A Hands-On Guide, Part 1*](#)

(March 1999)

[Advanced Programming for the Java™ 2 Platform](#)

(November 1999)

[Enterprise JavaBeans™ Tutorial](#)

(February 1999)

[Java 2D™: Styled Text](#)

(September 1998)

[Management](#)

(May 1999)

[Fundamentals of JFC/Swing: Part II](#)

(April 1999)

[Fundamentals of JFC/Swing: Part I](#)

(March 1999)

[Fundamentals of Java™ Servlets](#)

(January 1999)

[Fundamentals of Java Security](#)

(November 1998)

[JavaBeans™ Short Course](#)

(October 1997)

[JDBC™ Short Course](#)

(October 1997)

Tutorials by Category

[Downloads](#)

If you find it easier to download a tutorial or short course and work offline, look here for downloadable zip and PDF files.

Java Platform Programming

[The Java Tutorial \(Java Series\)](#)

[Essentials of the Java™ Programming Language: A Hands-On Guide, Part 1](#)

[Essentials of the Java™ Programming Language: A Hands-On Guide, Part 2](#)

[Advanced Programming for the Java™ 2 Platform--JDC](#)

Note: [The Java Tutorial](#), [Essentials of the Java Programming Language](#) and [Advanced Programming for the Java 2 Platform](#) are available as books from online book sellers.

[Beans](#)

JavaBeans™, Part II (Updated, November 2000)

JavaBeans™, Part I (Updated, October 2000)

Introduction to the JavaBean™ API
JavaBean™ Short Course
Enterprise JavaBean™ Tutorial

[Collections](#)

Introduction to the Collections Framework

[Distributed Computing](#)

JDBC™ 2.0 Fundamentals Short Course
JavaServer Pages Fundamentals
Protocol Handlers
Introduction to CORBA
JDBC™ Short Course, JDK 1.1
Fundamentals of RMI
Fundamentals of Java™ Servlets

[Graphics and Media](#)

Java 3D™ API
Java™ Advanced Imaging
Java 2D™: Styled Text

[Graphical User Interfaces \(GUIs\)](#)


AWT Fundamentals
The Swing Tutorial
Fundamentals of JFC/Swing: Part I-Part II
Effective Layout Management

[Java 2 Enterprise Edition](#)

Writing Enterprise Applications for the Java™ 2 Enterprise
Edition Reference Implementation

[Security](#)

Fundamentals of Java Security

Printable Page 

[This page was updated: 8-Jun-2001]

[Products & APIs](#) | [Developer Connection](#) | [Docs & Training](#) | [Online Support](#)
[Community Discussion](#) | [Industry News](#) | [Solutions Marketplace](#) | [Case Studies](#)

[Glossary](#) | [Feedback](#) | [A-Z Index](#)

For more information on Java technology
and other software from Sun Microsystems, call:
(800) 786-7638
Outside the U.S. and Canada, dial your country's
[AT&T Direct Access Number](#) first.



Copyright © 1995-2001 [Sun Microsystems, Inc.](#)
All Rights Reserved. [Terms of Use](#). [Privacy Policy](#).