# Forte™ for Java™, Internet Edition Tutorial

**Forte for Java, Internet Edition, 2.0**

Sun Microsystems, Inc.
901 San Antonio Road
Palo Alto, CA 94303
U.S.A. 650-960-1300

Part No. 806-7515-10
December 2000, Revision A

# Contents

# 3 Creating the Basic Tutorial Application

# Preface

Welcome to the Forte™ for Java™, Internet Edition tutorial! In this tutorial, you will learn how to use the features introduced in the Internet Edition, namely, support for Web applications that use Java™ Servlet and JavaServer Pages™ technology, and database access using Forte for Java custom tag libraries and Transparent Persistence.

**Who should read this book?** This tutorial creates a simple web application that interacts with a database and displays dynamically generated content. The design and architecture conforms to the Java™ 2 Platform, Enterprise Edition Blueprints resources. Anyone wanting to learn how to use the features of Forte for Java, Internet Edition, to build the components of a web application will benefit from working through this tutorial. Before starting it, you should be familiar with the following subjects:

- Java programming language
- Java Servlet syntax
- JDBC™ enabled driver syntax
- JavaServer Pages syntax
- HTML syntax
- Relational database concepts (such as tables and keys)
- How to use the chosen database

**Before you read this book:** The following list of resources can help you understand the concepts upon which this tutorial is based:

- Java™ 2 Platform, Enterprise Edition Blueprints — `www.java.sun.com/j2ee/blueprints`
- *Java™ 2 Platform Enterprise Edition Specification* — `www.java.sun.com/products`
- *Java™ Servlet Specification, v2.2* — `www.java.sun.com/products/servlet/index.html`
- *JavaServer Pages™ Specification, v1.1* — `www.java.sun.com/products/jsp/index.html`

# Organization of This Manual

This manual is designed to be read from beginning to end. Each chapter in the tutorial builds upon the code developed in earlier chapters.

The following table briefly describes the contents of each chapter:

| Chapter | Description |
|---|---|
| Chapter 1, "Getting Started" | Describes the software requirements for the tutorial, explains how to install the tutorial database table, and shows how to start the Forte for Java development environment, if you have not done so already. It also includes a descriptive list of the installed Forte for Java directories. |
| Chapter 2, "Introduction to the Tutorial" | Describes the architecture of the tutorial application. |
| Chapter 3, "Creating the Basic Tutorial Application" | Provides step-by-step instructions for creating the tutorial application, a simple online shopping cart application for the purchase of music CDs |
| Chapter 4, "Adding Transparent Persistence to the Tutorial Application" | Describes how to use Transparent Persistence to write customer data to a database when the customer of the tutorial application wishes to place an order. |

# Conventions

This table provides information about the conventions used in this document.

| Format | Description |
|---|---|
| *italics* | Italicized text represents a placeholder. Substitute an appropriate clause or value where you see italicized text. Italicized text is also used to designate a document title, for emphasis, or for a word or phrase being introduced. |
| `monospace` | Monospace text represents example code, commands that you enter on the command line, directory, file, or path names, error message text, class names, method names (including all elements in the signature), package names, reserved words, and URLs. |
| **`monospace bold`** | Monospace bold text represents user input contrasted with computer output. |
| ALL CAPS | Text in all capitals represents file system types (GIF, TXT, HTML and so forth), environment variables, or acronyms (FFJ, JSP). |
| *Key+Key* | Simultaneous keystrokes are joined with a plus sign. For example, Ctrl+A means press both keys simultaneously. |
| *Key-Key* | Consecutive keystrokes are joined with a hyphen. For example, Esc-S means press the Esc key, release it, then press the S key. |

# Forte for Java, Internet Edition Documentation Set

Forte for Java offers a set of books delivered in Acrobat Reader (PDF) format and online help. This section provides descriptions of these documents.

## Documentation Set

You can download the following documents from the Forte for Java web site:

- The Forte for Java programming series:
  - *Introduction to the Programming Series*

    Introduces the two books in the Forte for Java, Internet Edition programming series.
  - *Building Web Components*

    Describes how to build a web application as a J2EE web module using JSP pages, servlets, tag libraries, and supporting classes and files.
  - *Programming Persistence*

    Describes support for different persistence programming models provided by Forte for Java: JDBC and Transparent Persistence.
- *Forte for Java, Internet Edition Tutorial*

  Provides step-by-step instructions for building a simple web application using tools introduced in Forte for Java, Internet Edition, which facilitate creating a web module, as described in the *Java™ 2 Platform Enterprise Edition Specification*.

## Online Help

Online help is available inside the Forte for Java development environment. You can access it by pressing the help key (Help on Solaris, F1 on Windows and Linux), or by choosing Help > Contents from the Help menu. This displays a list of help topics and a search facility.

## Javadoc

Javadoc documentation is available within the IDE for many Forte for Java modules. Refer to the Release Notes for instructions for installing this documentation. When you start the IDE, you can access this Javadoc documentation within the Javadoc pane of the Explorer.

# Chapter 1

# Getting Started

This chapter explains what you need to do before starting the Forte for Java, Internet Edition tutorial. It also duplicates some installation information from the *Forte for Java Installation Guide*. The topics in this chapter are:

# Software Requirements for the Tutorial

This section describes how to prepare your system before starting the Forte for Java, Internet Edition tutorial. This means making sure you have everything required to run the Forte for Java integrated development environment (IDE), as well as what is required in addition to create and run the tutorial.

## What You Need to Run the Forte for Java IDE

Ensure that your system has been installed according to the instructions in the *Forte for Java Installation Guide*. It is important that you are running the proper version of the Java Development Kit. The installer will search for JDK™ software and install the correct version, if it is not already installed.

## What You Need to Create and Run the Tutorial

Everything that you need to create and run the tutorial is included in the default installation of Forte for Java, Internet Edition installation. This includes:

- Database software: PointBase Embedded Server, version 3.4

  To see if PointBase is installed with the Forte for Java IDE you are using, look for a `pointbase` directory under the Forte for Java home directory (referred to in this document as *`forte4j_home`*). If PointBase was not installed, run the installer again to install it.

- A web server

  Tomcat 3.2 beta 4 from the Jakarta Project of the Apache Software Foundation, is a Java Servlet 2.2 and JavaServer Pages 1.1 reference implementation. When bundled with Forte for Java, it provides the functionality of a web server for testing purposes. Currently, this is the only tool you can use with Forte for Java for this purpose. In future, other web servers will qualify.

- A web browser: ICE Browser 4.0 from Integrated Systems, Inc.

You can configure Forte for Java to use an alternate database or web browser. The next two sections describe how to do this.

## Using Alternate Database Software

If you do not want to use the PointBase database with the tutorial, you must have at least client database software installed on your system, as well as a JDBC-enabled driver. You must also manually put a copy of the driver into the `lib/ext` directory under the Forte for Java installation directory.

Versions of Oracle and Microsoft SQLServer and their appropriate JDBC-enabled drivers are supported in Forte for Java, Internet Edition. For the supported versions, check the Support Matrix section of the *Release Notes*, which are accessible from the Forte for Java web site.

### How to Specify Which Database to Use

To tell Forte for Java that you are using PointBase or any other database, you simply specify the correct values for your type of database when you create a JDBC connection string. This is explained in the tutorial.

## Using Alternate Web Browsers

When you run a JSP page, servlet, or HTML page, Forte for Java automatically starts a web browser to display the results. You can use any web browser, such as Netscape or Microsoft Internet Explorer, instead of the default ICE Browser.

### How to Specify Which Web Browser to Use

You configure web browsers by setting the Web Browser property of the global JSP option, as follows.

➤ **To set the Web Browser global property:**

1. Start Forte for Java and choose Tools > Global Options to bring up the Global Options dialog box.

   See "Starting the Forte for Java Development Environment" on page 18.

2. Select JSP in the list of option categories in the left pane of the window to display JSP properties.
3. Click on the Web Browser value to display the file browser ("...") button, then click that to display the Browser Properties dialog box.
4. In the Browser Properties dialog box, either set the Internal or External option.

   If you set External, use the file browser ("...") button to find the executable of the web browser you want to use.

5. Click **OK** to apply the change and dismiss the dialog box.

# Installing the Tutorial Database Table

Before you start the Forte for Java, Internet Edition tutorial, you must create and install several database tables in a database of your choice. SQL scripts for several versions of SQL are provided to create these tables.

These scripts are found in the *forte4j_home*/Development/tutorial/CDShopCart/SQLscripts directory. They are:

| Script name | Description |
|---|---|
| CDCatalog_pb.sql | Creates and populates tables used by the tutorial in PointBase SQL format. |
| CDCatalog_ora.sql | Creates and populates tables used by the tutorial in Oracle SQL format. |
| CDCatalog_ms.sql | Creates and populates tables used by the tutorial in SQLServer SQL format. |

The CDCatalog scripts create the following database schemas.

**Table 1** *CDCatalog Database Tables*

| Table Name | Columns | Primary Key | Other |
|---|---|---|---|
| CD | id | yes | |
| | cdtitle | | |
| | artist | | |
| | country | | |
| | price | | |
| Sequence | tableName | yes | |
| | tnextPK | | |
| CdOrder | id | yes | |
| | orderDate | | |
| OrderItem | orderID | yes | |
| | lineItemID | | Foreign key, references CdOrder (id) |
| | productID | | Foreign key, references CD (id) |

The CD table has the following records inserted:

| ID | CDtitle | Artist | Country | Price |
|---|---|---|---|---|
| 1 | Yuan | The Guo Brothers | China | 14.95 |
| 2 | Drums of Passion | Babatunde Olatunji | Nigeria | 16.95 |

| ID | CDtitle | Artist | Country | Price |
|---|---|---|---|---|
| 3 | Kaira | Tounami Diabate | Mali | 13.95 |
| 4 | The Lion is Loose | Eliades Ochoa | Cuba | 12.95 |
| 5 | Dance the Devil Away | Outback | Australia | 14.95 |

The Sequence table has the following records inserted:

| tableName | nextPK |
|---|---|
| CdOrder | 1 |

## Installing the Tables in a PointBase Database

The instructions given here first create a "tutorial" database, and then load the tables into it, but you can install the tables in any PointBase database you choose.

➤ **To install the tutorial database tables in the PointBase database:**

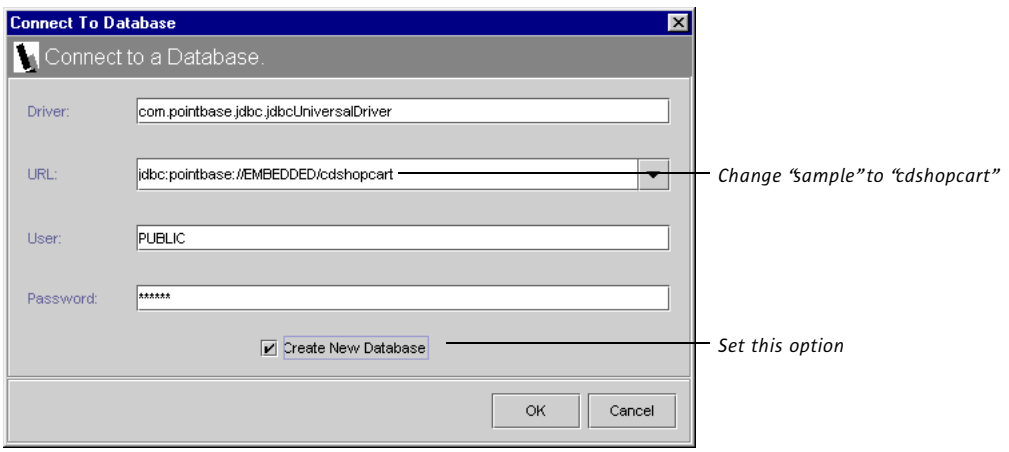
1 Start the PointBase Console, which is in the *forte4j_home*\pointbase directory.

**On Solaris or Linux:** Run the `console.sh` file.

**On Windows:** Double-click on the `console.bat` file, or choose Start > Programs > Forte(tm) for Java(tm), release 2.0, Internet Edition > PointBase-Embedded > pbconsole.

The Connect To Database dialog box appears, showing values for the PointBase driver to the default sample database.

2 Change the word "sample" at the end of the URL field to **cdshopcart**, as shown.




3 Toggle the Create Database option to on and click OK.

The PointBase console is displayed. After a number of status messages about creating the new database, a message ending in "Ready" is displayed.

4. Choose File > Open to display a file browser dialog box.
5. Use the file browser to find the `CDCatalog_pb.sql` file and click Open.

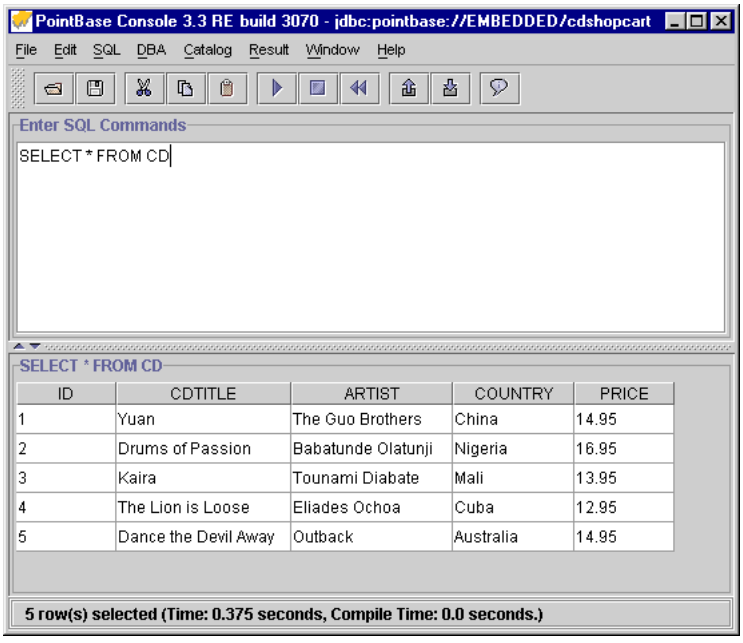
   The contents of the `CDCatalog_pb.sql` file are copied to the SQL entry window.
6. Choose SQL > Execute All.

   The message window confirms that the script was executed. (Ignore the initial messages beginning "Cannot find the table..." These appear because of the DROP statements for each of the tables that have not been created yet. These DROP statements will be useful in the future if you want to rerun the script to initialize the tables.)
7. Test that you have created the table by clearing the SQL entry window (Window > Clear Input) and typing:

```
SELECT * FROM CD
```

8. Select the statement and choose SQL > Execute.

   Your console should display the CD table.




9. Close the PointBase Console window.

## Installing the Table in Other Databases

Consult the support matrix in the *Forte for Java Installation Guide* for which databases, their versions, and their corresponding JDBC drivers are supported. Make sure there is a copy of the appropriate JDBC-enabled driver in the *forte4j_home*\lib\ext directory.

This section provides methods for installing the tutorial table on an Oracle and Microsoft SQLServer database.

➤ **To install the tutorial database in other supported databases:**

- **For Oracle:** Use file indirection with SQL*Plus to load the CDCatalog_ora.sql script.

  For example, to load the script into a database on Service Name TUTORIAL with user/password of tutorial/tutorial, enter the following on the command line:

  ```
  c:\>cd forte4j_home\Development\tutorial\CDShopCart\SQLscripts
  c:\>sqlplus tutorial\tutorial@TUTORIAL @CDCatalog_ora.sql
  ```

- **For Microsoft SQLServer:**

  **a** Create a database or designate an existing one.

  **b** Modify the first line of the CDCatalog_ms.sql file.

  ```
  use CDShopCart
  ```

  to change "CDShopCart" to the name of your database.

  **c** Use file indirection to load the CDCatalog_ms.sql script.

  For example, to load the script into a database on server MY_MSSQL with user/password of jane/jane, type the following command:

  ```
  c:\>cd forte4j_home\Development\tutorial\CDShopCart\SQLscripts
  c:\>isql -SMY_MSSQL -Ujane -Pjane < cdcatalog_ms.sql
  ```

# Starting the Forte for Java Development Environment

Starting Forte for Java means simply running the program executable, which is described in the *Forte for Java Installation Guide*. The descriptions here are provided for your convenience.

## Single-User and Multiuser Modes

You can use the Forte for Java development environment in two modes: single-user or multiuser. Multiuser mode is designed for development groups who need to synchronize their development activities. To this end, multiuser mode requires a user who acts as the administrator for maintaining, upgrading, and installing new modules.

You can use either single-user or multiuser mode for creating the tutorial. However, single-user is recommended, because it is easier to set up.

## Starting Forte for Java on Solaris™ 7/8, Linux Redhat 6.2, and other UNIX™ Software

After installation, a `runide.sh` script is in *`forte4j_home`*`/bin` directory. Launch this script by typing:

```
$ sh runide.sh
```

## Starting Forte for Java on Microsoft Windows

After installation, there is a Forte for Java, Internet Edition 2 icon on your desktop. Double-click this icon to start the IDE. This icon is one of several available, depending on which mode you want to run the IDE in. For example, the four modes available are represented in the *`forte4j_home`*`\bin` directory as the following executable files:

- `runidew.exe` — launches Forte for Java without a console window. This launcher is used when you launch the IDE from the shortcut on the Desktop or Start menu.
- `runide.exe` — launches Forte for Java with a console window that includes standard error and standard output from the IDE. On the console, you can press Ctrl-Break to get a thread dump or Ctrl-C to immediately terminate the program.
- `runidew_multiuser.exe` — launches a multiuser version of Forte for Java without a console window. When you first run this executable, you are prompted to enter a directory where your files will be stored. This information will be placed in your Windows registry, so that the directory you specify in the prompt will be used whenever you launch the IDE in the future.
- `runide_multiuser.exe` — launches a multiuser version of Forte for Java with a console window.

Alternatively, you can launch Forte for Java by choosing Start > Programs > Forte for Java, Internet Edition 2 > Forte for Java, Internet Edition 2. Finally, you can run any of the executables from the command line. For example:

```
C:\> runide.exe
```

## Command-Line Switches

This section describes the switches that you can use to modify how you launch Forte for Java. This information is also available from the *Forte for Java Installation Guide*, but is provided here for your convenience.

On Windows machines

You can set options when running the IDE on the command line or by modifying the `ide.cfg` file that is in the `bin` directory of your installation directory. If you use the `ide.cfg` file, you can break the options into multiple lines. The loader tries to read this file before it starts parsing the command line options. This means that Java application options can be put in this file.

On Solaris, Linux, and other UNIX machines

You can modify the `ide.sh` file in the `bin` subdirectory of the installation directory, or you can create your own shell script that calls `ide.sh` with options.

**Table 2** *runide Command-Line Switches*

| Switch | Meaning |
|---|---|
| `-jdkhome` *`jdk_home_dir`* | Use the specified Java 2 SDK instead of the default SDK. |
| `-hotspot` | Use the HotSpot JVM (default). |
| `-classic` | Use the classic JVM. |
| `-cp:p` *`addl_classpath`* | Add a class path to the beginning of the Forte for Java class path. |
| `-cp:a` *`addl_classpath`* | Add a class path to the end of the Forte for Java class path |
| `-J`*`jvm_flags`* | Pass the specified flag directly to the JVM. (There is no space between `-J` and the argument.) |
| `-ui com.sun.java.swing.plaf.windows.WindowsLookAndFeel` | Run the IDE with the Windows look and feel. (There is no break between `windows.` and `WindowsLookAndFeel`.) |
| `-ui com.sun.java.swing.plaf.motif.MotifLookAndFeel` | Run the IDE with the Motif look and feel. (There is no break between `motif.` and `MotifLookAndFeel`. |
| `-fontsize` *`fontsize`* | Set the font size used in the GUI to the specified size. |
| `-userdir` *`user_directory`* | Use the specified directory for configuration files. |
| `-h` or `-help` | Open a GUI dialog box that lists the command-line options. |

## Exiting Forte for Java

Exit Forte for Java by choosing File > Exit.

# Forte for Java, Internet Edition Directory Structure

This section describes the Forte for Java, Internet Edition directory structure. This information is also available from the *Forte for Java Installation Guide*, but is provided here for your convenience.

When you install Forte for Java on your machine, the following subdirectories are included in your installation directory.

**Table 3** *Forte for Java Directory Structure*

| Directory/Folder | Purpose |
|---|---|
| `beans` | Contains JavaBeans™ components installed in Forte for Java. |
| `bin` | Includes Forte for Java launchers (as well as the `forte4j.cfg` file on Windows installations) and `addtopath.bat`, which is used by the batch file startup. |
| `Development` | The directory mounted by default in Filesystems. Objects you create in Forte for Java will be saved here unless you mount other directories and use them instead. |
| `Development/examples` | Contains several example applications. |
| `Development/tutorial` | Contains several tutorial applications, including the CDShopCart tutorial described in this document and its database scripts. |
| `docs` | Contains the Forte for Java help files and PDF files of the documents in the Forte for Java programming series, as described in "Documentation Set" on page 10, including this tutorial. (Release Notes are found under *`forte4j_home`*.) |
| `javadoc` | The directory mounted by default in Forte for Java Javadoc repository. Javadoc supplied with the IDE and Javadoc documentation that you create in Forte for Java is stored here. |
| `lib` | Contains JAR files that make up Forte for Java's core implementation and the open APIs. |
| `lib/ext` | Contains extensions to Forte for Java for things such as JavaHelp, Absolute Layout, javac, and regular expressions. Also includes all classes for PointBase (if installed). If you use a different database, you must copy the JDBC driver to this directory. |
| `lib/patches` | Any JAR or ZIP file included in this directory is automatically included at the beginning of the IDE's startup class path. That is, it will be a patch against the core. |
| `modules` | Any JAR file in this directory is a Forte for Java module. |
| `modules/ext` | Contains libraries used by modules. |
| `pointbase` | Contains the executables, classes, databases, and documentation for the PointBase Embedded database (if installed). |
| `sources` | Sources for libraries that may be redistributed with user applications. |

**Table 3** *Forte for Java Directory Structure (Continued)*

| Directory/Folder | Purpose |
|---|---|
| `system` | Includes files and directories used by Forte for Java for special purposes. It includes `ide.log`, which provides information useful when seeking technical support, and `project.last`, which contains information on Forte for Java projects. It is mounted as a hidden file system. |
| `system/Actions` | Contains actions that appear in Global Options under Actions. |
| `system/applet` | Policy file for debugging applets. |
| `system/Bookmarks` | Contains web bookmarks. |
| `system/ParserDB` | Holds databases that are used for Java code completion and other Editor functions. |
| `system/Startup` | Holds classes that are run at Forte for Java startup. |
| `teamware` | Contains Forte for Java TeamWare module files (if installed). |
| `teamware/bin` | Contains Forte TeamWare executable files. |
| `teamware/doc` | Contains Forte TeamWare documentation. |
| `teamware/readme` | Contains important late-breaking information about Forte TeamWare. |
| `temp` | Contains information used by the internal Tomcat server, the transaction monitor, and TeamWare. |

# Chapter 2

# Introduction to the Tutorial

In the process of creating the tutorial example application, you will learn how to use features that are introduced in Forte for Java, Internet Edition for building components of a web application.

To better understand the tools you will learn to use, this chapter first describes the application you will build, laying out its requirements and then describing an architecture that will fulfill those requirements. The final section describes how you use the Forte for Java, Internet Edition features—web module constructs, Forte for Java tag libraries, and Transparent Persistence—to create the application.

This chapter is organized into the following sections:

# Functionality of the Tutorial Application

The tutorial example application, CDShopCart, is a simple online shopping cart application for the purchase of music CDs. The customer uses a web browser to interact with the application's interface.

The application presents the customer with a catalog of products. From this catalog,:

1. The customer selects items to be added to a shopping cart
2. The customer can add more items to or remove existing items from the shopping cart.
3. When the customer is ready to buy what is in the shopping cart, the application writes the order to the database, displays a message that thanks the customer and provides a customer order number, and ends the session.
4. The customer can then exit the application or use a button on the order page to start a new shopping session.

## Application Scenarios

The interaction of the CDShopCart application begins with the customer's visit to the application's home page and ends when the customer either completes an order or quits the site. The scenarios that follow describe a few of the user's interactions with the CDShopCart application. Walking through these scenarios illustrates the requirements of the application, as well as interactions that happen within the application.

1. The customer starts the application by pointing the browser to the URL of the application's home page.

   The home page is the CD Catalog List page, which displays the list of available music CDs and their associated information: title, the CD id number, the name of the performing artist, the artist's country, and the price.

2. The customer selects a CD for purchase by clicking the Add button associated with a CD.

   This action causes the application to display a Shopping Cart page showing the selected CD title, its ID number, and price.

3. The customer selects more CDs for purchase by clicking the Resume Shopping button on the Shopping Cart page.

   The application redisplays the CD catalog page, so that the customer can select another CD, as in #2. The customer can repeat this sequence as many times as she likes, even adding the same CD multiple times (which adds more rows of the same CD to the cart).

4. The customer removes an item from her shopping cart by clicking the Delete button associated with the item on the shopping cart page.

   Clicking this button redisplays the shopping cart minus the item, unless she deleted the last CD in the cart. When the last CD is removed, a page is displayed that announces that the cart is empty.

5 The customer can click the Resume Shopping button on the page to return to the CD Catalog List page, or the Cancel Order button to end the session. (See #6 for the Cancel Order page.)

6 The customer decides to make a purchase and clicks the Place Order button on the Shopping Cart page.

   This action writes the order to the database, displays a "Thank You" message page, and ends the session. The customer can click the Resume Shopping link on the message page to start another session, or leave the application by closing the browser or going to a different URL.

7 The customer cancels an order by clicking the Cancel Order button on the shopping cart page.

   This causes the application to display a "cancel order" message page that ends the session. The Cancel Order page includes a Resume Shopping button, in case the customer wants to start a new session.

## Application Functional Specification

Given the kinds of scenarios in which the CDShopCart application would be used, the following items list the main functions for a user interface of an application that supports these shopping interactions.

- A set of links to navigate from page to page
- A master view of all the site's offerings through a displayed list
- A view of items selected for purchase
- Buttons on the catalog page for adding each item
- Buttons on the shop cart page for removing items
- A button on the shop cart page to initiate checkout
- A button on the shop cart page to cancel the order
- A button on the checkout page to return to the home page to begin a new order
- A button on the empty cart page to return to the home page
- A button on the empty cart page to cancel the order

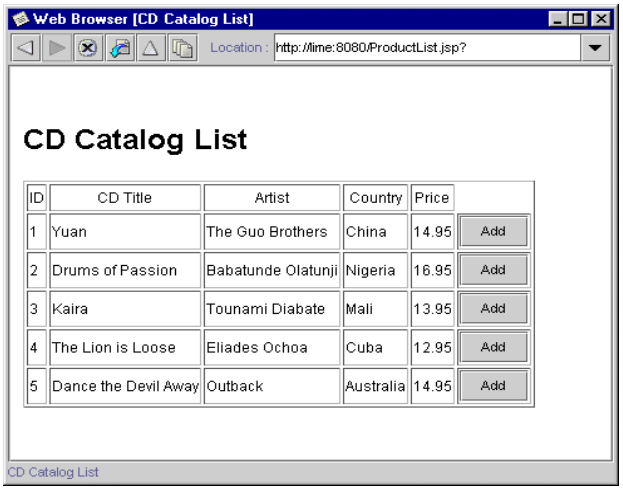
# User's View of the Tutorial Application

This section describes the user's view of the application, illustrating how the scenarios and the functional specification, described in "Functionality of the Tutorial Application" on page 24 are realized.

➤ **To run the CDShopCart application:**

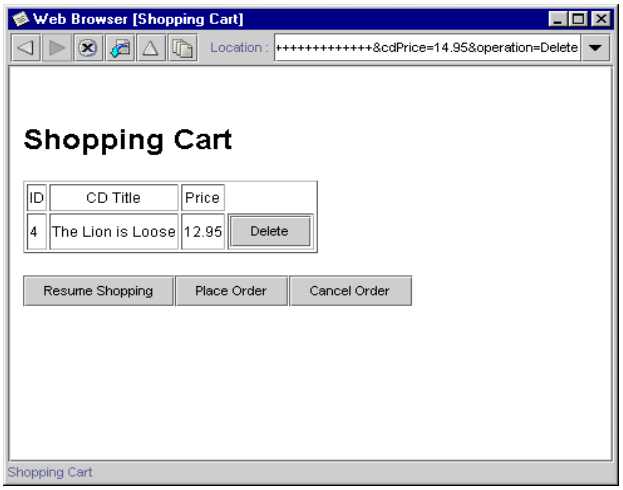
1 The application starts with a CD Catalog List page that displays a list of CD titles.

This page is created with the ProductList JSP page.




2 To add a CD to your shopping card, click the Add button in the row of that CD.

This action displays the Shopping Cart page with the selected CD on it. This page is created by the ShopCart JSP page.



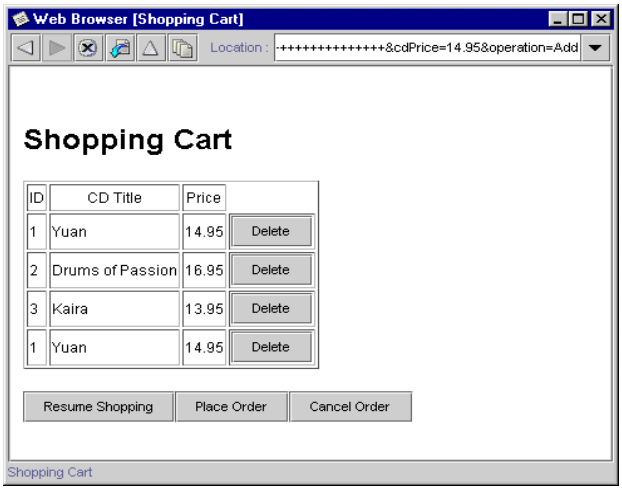
3 To add another CD, click Resume Shopping, which takes you back to the CD Catalog List page.

4 Click Add on the same or a different CD.

The Shopping Cart page is redisplayed with an additional selection.

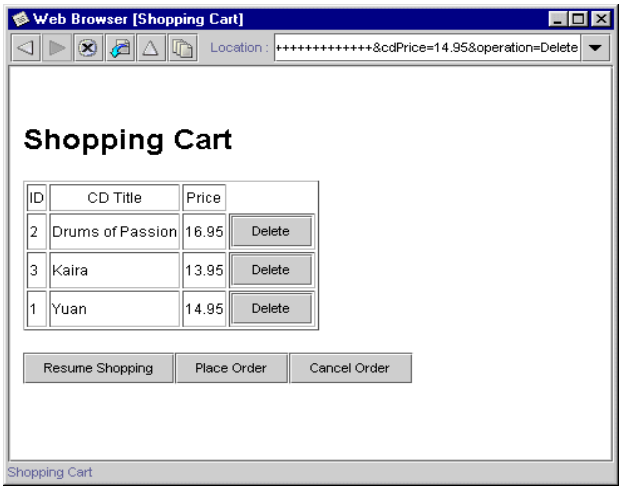
5 Repeat Step 2 and Step 3 until you have all the CDs you want to purchase.

The Shopping Cart displays your items. If you have chosen more than one of the same item, these are displayed as separate rows.

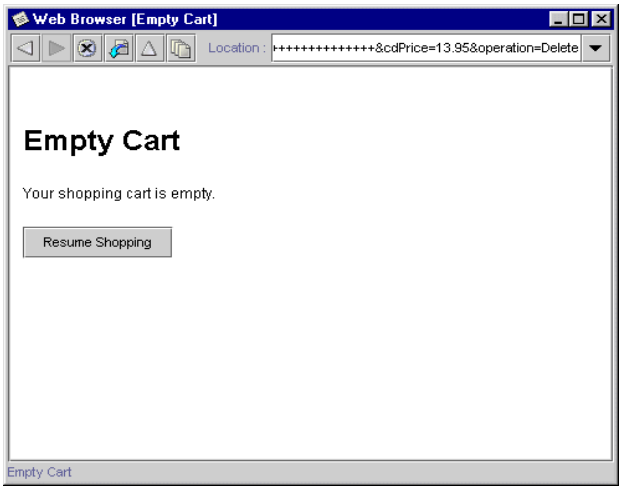



6 To remove an item, click the item's Delete button.

The table is redisplayed minus the removed item.



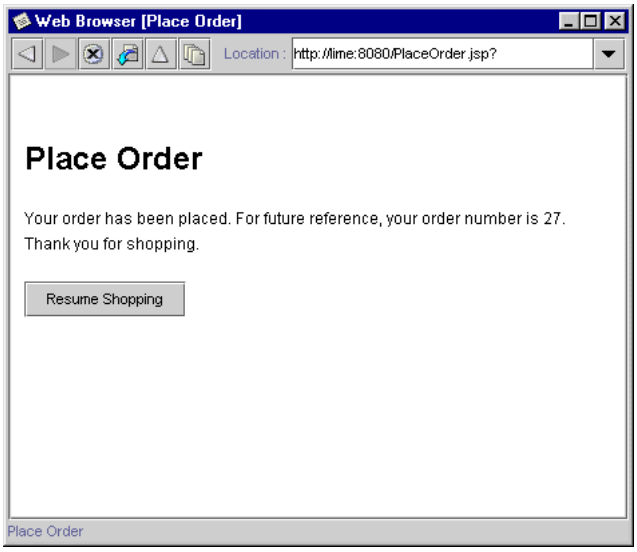
If you remove the last item in the table, the Empty Cart HTML page is displayed:




7 Click the Resume Shopping button to return to the CD Catalog List page.

8 To place an order, click the Place Order button on the Shopping Cart page.

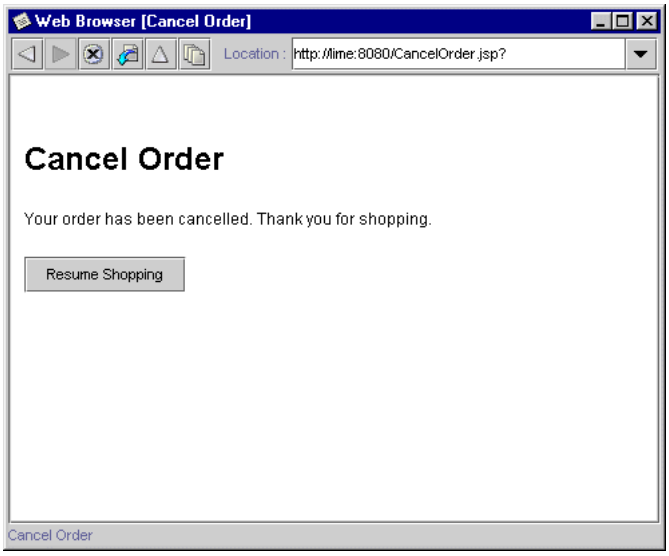
The Place Order page is displayed. This page is created by the PlaceOrder JSP page.




You can either exit the application by pointing the browser at a different URL, or start a new session by clicking the Resume Shopping button.

9 Or, to cancel your order, from the Shopping Cart page, click the Cancel Order button.

The Cancel Order page is displayed. This page is created by the CancelOrder JSP page.




You can start a new session by clicking the Resume Shopping button.

# Architecture of the Tutorial Application

The previous sections have given you an understanding of what the CDShopCart application is supposed to do. This section describes an architecture that supports those requirements.

The CDShopCart application is a web-centric application that uses a web client to send requests to and receive results from a web application. A *web application* is a bundle of web components and their supporting classes, beans, and files. *Web components* are server-side J2EE components, such as Servlets and JSP pages.

The CDShopCart application consists of one web module. A *web module* is the smallest deployable and usable unit of web resources in a J2EE application. A feature introduced in Forte for Java, Internet Edition is the web module construct, which automatically creates the required directory structures, default versions of required data objects, and other special services required by the web module.

For more information on web modules and related concepts, see *Building Web Components*. For information specific to the web module construct, see the Developing Web Modules section under JavaServlet Pages and Servlets in online help.

Figure 1 shows the CDShopCart application elements and their relation to one another.

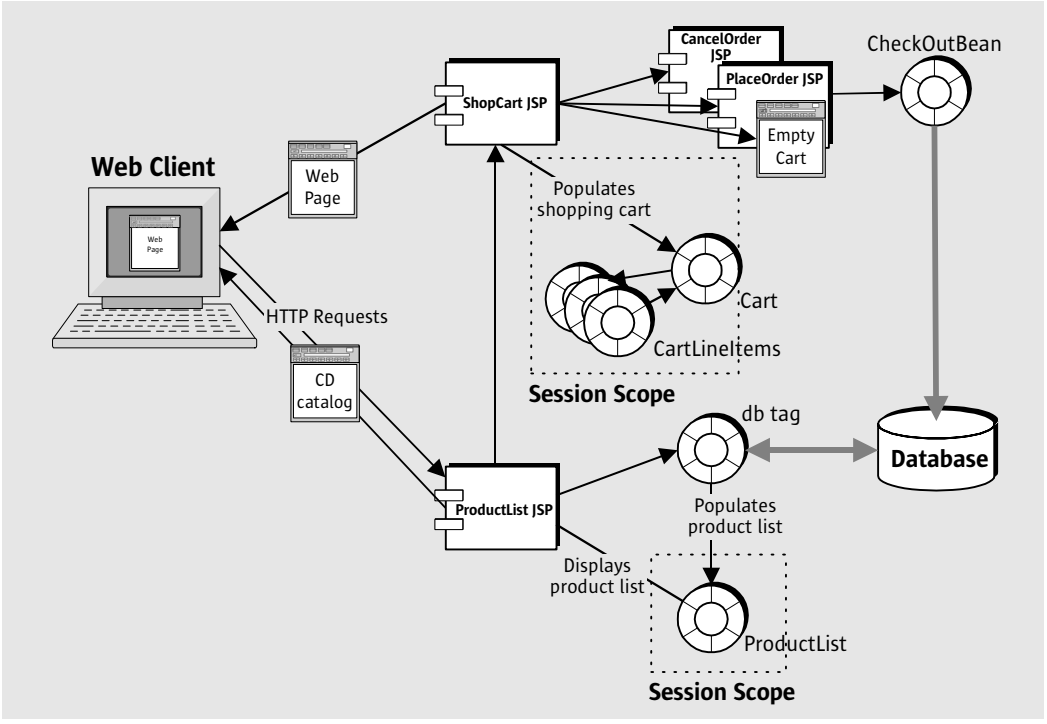



**Figure 1** *Architecture of the CDShopCart Application*

## Application Elements

Briefly, the elements shown in Figure 1 are:

- The *client* component

  The client component is a web browser that displays the application pages.

- The *service* component (a web module) that includes:
  - A ProductList JSP page that retrieves the product data from a database and displays it in a table on the CD Catalog List web page. ProductList also provides an Add button by which a user can add a CD to the shopping cart.
  - A ShopCart JSP page that displays CDs selected for purchase in a table on the Shopping Cart web page, and provides Delete, Place Order, Cancel Order, and Resume Shopping buttons.
  - An EmptyCart HTML page that displays a message when the customer deletes the last item in the shopping cart. This JSP page includes a Resume Shopping link.
  - A CancelOrder JSP page that displays a message that the order has been canceled and a Resume Shopping button for returning to the ProductList JSP page.
  - A PlaceOrder JSP page that uses CheckOutBean to save the order to the database. PlaceOrder displays a message that the order has been placed, and includes a Resume Shopping button to the ProductList JSP page.
  - A Cart bean that represents the items selected for purchase.
  - A CartLineItem bean that represents a cart line item.
  - A CheckOutBean bean that uses Transparent Persistence to update the database with the order data.

## Service Component Details

The service component of the CDShopCart application is a web module that includes four JSP pages that coordinate the application's behavior, given input from the client, and supporting elements to the web module include JavaBeans elements and an HTML page file.

**ProductList JSP page** This page locates the session for the current user or creates one if it doesn't exist. This page uses tags from the Forte for Java database tag library to access the list of CDs from the database and other tags from the presentation library to display them in a table. This page also provides an Add button on each CD line item it displays.

**ShopCart JSP page** When the user clicks the Add button on the ProductList page, the data of the line item is passed to this JSP page, which instantiates a Cart object consisting of CartLineItem objects, and uses tags from the Forte for Java presentation tag library to display it in a table. This page provides a Delete button on each cart item. When the user clicks this button, the page uses a scriptlet to remove the item, update the table data, and redisplay it. If the item removed is the last item in the cart, this page forwards to the EmptyCart page.

This page also provides Resume Shopping, Cancel Order, and Place Order buttons. These buttons forward to the ProductList page, the CancelOrder page, and the PlaceOrder page, respectively.

**Cart JavaBean** This bean has a lineItems attribute and includes the methods for getting and removing the CartLineItem objects. This bean is imported by the ShopCart page.

**CartLineItem Javabean** This bean has CD related attributes, and includes methods for getting and setting attributes of Cart line items (ID, title, artist, country, and price).

**CancelOrder JSP** This JSP page is called when the user clicks the Cancel Order button on the ShopCart page. This page invalidates the session, displays a "Your order is canceled" message, and provides a Resume Shopping button to the ProductList page.

**EmptyCart JSP** This JSP page is called when the user clicks the Place Order button on the ShopCart page when the number of line items is zero. It displays a message that the cart is empty and includes a Resume Shopping button.

**PlaceOrder JSP** This JSP page is called when the user clicks the Place Order button on the Cart page when there are items in the cart. It uses the CheckOutBean to save the order to the database, then invalidates the session, and displays a "Thank you for your order" message. PlaceOrder includes a Resume Shopping button.

**CheckOutBean JavaBean** This bean uses Transparent Persistence tools to connect to the database and associate a unique order number with the customer's order, then update the database with the new order data.

# Overview of Tasks for Creating the Tutorial Application

The tutorial is divided into two chapters. In the first, you create the basic application, but without saving the order to the database. In the second chapter, you learn how to save the order to the database with Transparent Persistence.

Before you can create the tutorial application, you must have Forte for Java installed and set up to run, and the tutorial database tables installed, as described in Chapter 1, "Getting Started."

## Creating the Basic Application

In Chapter 3, "Creating the Basic Tutorial Application," you learn how to use the following Forte for Java features:

- Web modules (creating, developing, and test running)
- Provided database tags for connecting to and interacting with a database
- Provided presentation tags for iterating through the retrieved data

In addition, you create the supporting elements: several beans and an HTML page.

## Creating a Web Module

Forte for Java provides a tool for automatically creating the hierarchical directory structure of a web application. This tool is the web module. Except for elements relating to Transparent Persistence development, you develop the entire CDShopCart application within a single web module construct.

The tutorial doesn't try to provide complete information about how to develop a web module. The section "Creating a Web Module" on page 38 serves as an introduction to the subject, outlining the basic elements of the structure, and the (very easy) method for creating it. When you want to know more about how to develop web modules, see *Building Web Components*.

## Using Forte for Java Tag Libraries

Within the CDShopCart web module, you create the ProductList JSP page to fetch the CD catalog data to display on the CD Product List web page. You then create the ShopCart JSP page to display CDs that the user selects for purchase. You use Forte for Java custom JSP tag libraries for database access and data presentation functions to accomplish this.

Database and presentation tags in ProductList

The section "Using Forte for Java Custom Tags" on page 42 describes how to use the custom database tags to make the JDBC connection to the database and fetch the CD data. It then describes how to use presentation tags to iterate through the resulting data, so that ProductList can display it in an HTML form on the web page.

Presentation tags in ShopCart

The ShopCart JSP page displays CD data passed to it by the ProductList JSP page. The section "Add Code to Add or Remove an Item From the Shopping Cart Table" on page 59 demonstrates how to use presentation tags to iterate through the passed values to find the individual field values, so they can be displayed in the correct columns in the cart table.

## Creating the Supporting Elements

The supporting elements for the ShopCart JSP page are two beans (Cart and CartLineItem), two JSP pages (CancelOrder and PlaceOrder), and one HTML page (EmptyCart).

Beans that hold cart item data

The section "Create the CartLineItem JavaBeans Component" on page 54 shows you how to create a bean whose object holds the parameters of a line item passed to ShopCart from ProductList when a user clicks the Add button on the item. Then, in "Create the Cart JavaBeans Component" on page 57, you learn how to create a bean whose object holds the accumulated line items that have been selected. The Cart bean has methods for removing line items from the cart.

An HTML error page

In "Empty Cart Page" on page 64 section, you create an HTML page that displays a message that the cart is empty. This is to avoid displaying an empty form on the Shopping Cart page. The exercise teaches you how to open an HTML source file in the IDE, which is different from opening other source files.

JSP pages that display messages appropriate to button actions

You create two more JSP pages, but these hold minor logic compared with ProductList and ShopCart. In "Place Order JSP Page" on page 65, you create a JSP page that thanks you for placing an order and then invalidates the session. It is this page that you will expand in the next chapter to use Transparent Persistence to write the order to a database. In "Cancel Order JSP Page" on page 67, you create a similar page that announces that your order is canceled and invalidates the session.

## Test Running the Application

Throughout this chapter, you test run each element just after you create it. Forte for Java automatically deploys a web module to its internal container when you execute one of the web module's components.

# Adding Transparent Persistence

In Chapter 4, "Adding Transparent Persistence to the Tutorial Application," you learn to use Transparent Persistence to write the order to the CdOrder table you created in Chapter 1, "Getting Started."

## Creating the Persistence-Capable Classes

You use three of the tutorial database tables—CdOrder, OrderItem, and Sequence—to generate unique numbers for the customer's order and the line items in that order. This allows you to store individual orders uniquely. To be able to operate on these database tables as Java classes, you capture the database schema, then generate persistence-capable classes from them. The final step in making the classes persistent is to "enhance" them, which is done by the process of packaging them in a JAR file. You then put the JAR file within the application's web module hierarchy, so that it will be included in the web module's WAR (Web ARchive) file. These procedures are described in "Creating the Persistence-Capable Classes" on page 73.

## Saving the Order to the Database

A single bean (CheckOutBean) encapsulates all the Transparent Persistence procedures for writing the order data to the database when the user clicks the Place Order button. The procedures for creating this bean are described in "Creating the Persistence-Aware Bean" on page 82. The procedures include creating a Persistence Manager Factory, which creates a template for the database connection, and then creating a Persistence Manager, which manages the transaction and query operations. You use the methods of the Query interface to perform the required database functions that you would otherwise have to use SQL or some other data store-specific language to perform.

## Using the Results to Place the Order

In the section "Modifying the PlaceOrder Page to Call CheckOutBean" on page 89, you modify the PlaceOrder JSP page to use the results from CheckOutBeans methods to write the order to the database and display the returned order number.

## Test Running the Whole Application

You now run the CDShopCart application, add a few CDs to the shopping cart, and then place the order. This exercise demonstrates the power of Transparent Persistence by displaying a generated order number. You can also check this action by checking changes in your database table data.

## End Comments

The tutorial application is designed to be brief enough for you to create in a relatively short time (a day or so). This places certain restrictions on its scope. For example:

- There is no error handling.
- There are no debugging procedures.
- There is no description of how to create the WAR file for deployment.

Future releases will have these procedures.

Although the tutorial application is designed to be a simple application that you can complete quickly, you might want to import the entire application, view the source files, or copy and paste method code into methods you create. The CDShopCart application is accessible from within the IDE, under the *forte4j_home*/Development/tutorial file system.

# Chapter 3

# Creating the Basic Tutorial Application

This chapter describes, step by step, how to create the CDShopCart tutorial application, except for writing the order data to a database. Before you can create the tutorial application, you must have Forte for Java installed and set up to run, and the tutorial database tables installed, as described in Chapter 1, "Getting Started."

This chapter is organized under the following topics:



You test each component as you create it. By the end of this chapter, you will be able to run the basic application, as described in Chapter 2, "Introduction to the Tutorial," except that Step 9 will not display the order number. That part is covered in Chapter 4, "Adding Transparent Persistence to the Tutorial Application."

# Creating a Web Module

The CDShopCart application is a web-centric application. Web-centric applications are comprised of web modules. The CDShopCart application is a very simple application, composed of only one web module.

This section shows you how to set up your application as a web module by using the Forte for Java, Internet Edition web module construct.

## What Is a Web Module?

According to the *Java Servlet Specification, v2.2*, "a web application exists as a structured hierarchy of directories." The root of this hierarchy is the `document root`, which holds all the files that are part of the web application. The hierarchy also includes a special non-public subdirectory, the `WEB-INF` directory, for things that are related to the web application but are not to be served directly to the client. Items in the `WEB-INF` directory include the web deployment descriptor (the `web.xml` file), and servlet and utility classes used by the web application loader to load classes from.

No J2EE deployment construct forces you to develop your web application within this special directory hierarchy. However, your application's files must eventually be part of a web module structure in order to package them as a WAR file (a Web ARchive format file) for delivery into a web container. The Forte for Java web module construct automates much of the process of creating the required directory hierarchy, as well as filling it with default version of some of the objects.

**For more information** This tutorial does not try to provide complete information about developing web modules. For that task, see *Building Web Components*. Also, consult the Forte for Java online help for more details on web modules. Look for this information in the Developing Web Modules folder under the JavaServer Pages and Servlets main folder.

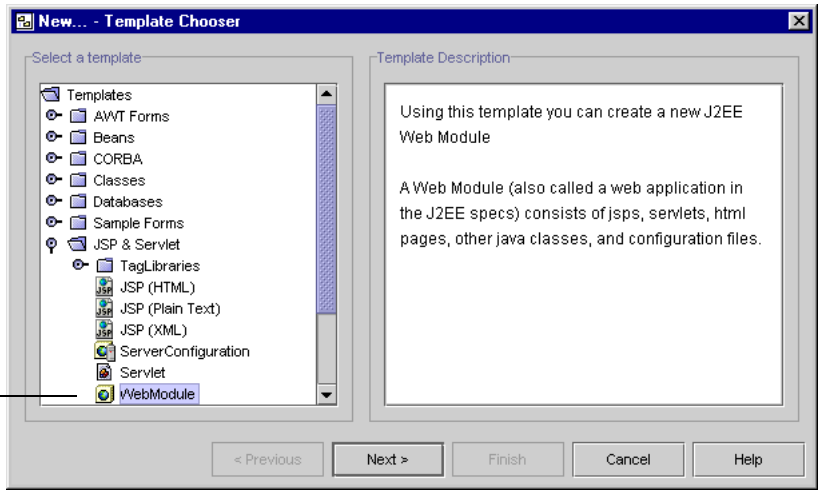
## Create the CDShopCart Web Module

In this section, you create the web module for the CDShopCart application. A web module is a directory. You can use the Forte for Java web module feature to convert an existing directory into a web module, but such a directory doesn't yet exist. You will create one from scratch.

➤ **To create the CDShopCart web module:**

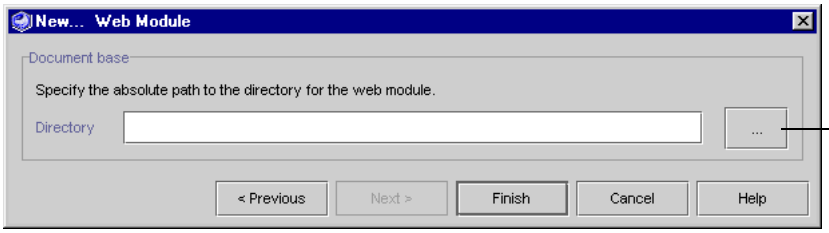
1. With the Filesystems pane of the Explorer window active, choose File > New to display the New Template Chooser dialog box.

2 Open the `JSP & Servlet` folder and select `WebModule`.




*WebModule*

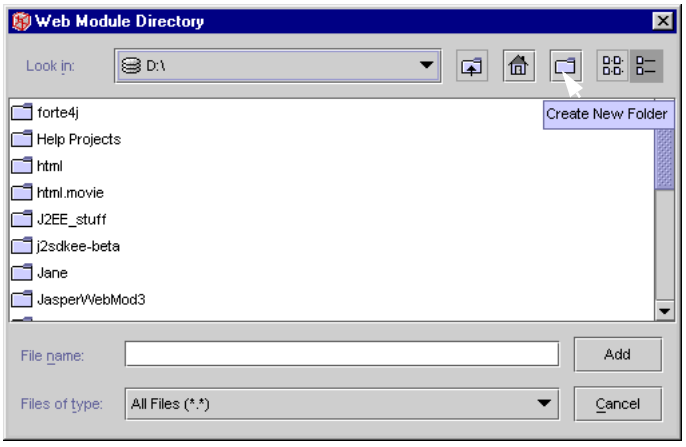
3 Click Next to display a dialog box for specifying the path to the directory for the web module.




*Click here to browse for an existing directory or add a new one*

4 Click the browse button ("...") to add a new directory.

5 On the Web Module Directory dialog box that appears, find the location you want and click the New Folder button.



6 Scroll down to find the `New Folder` folder.

7 Select the `New Folder` folder, and click on it again to make the name editable.

8 Replace the old name with **`CDShopCart`** and press **Enter.**

9 In the list of folders, find the `CDShopCart` folder, make sure it is selected, and then click Add.

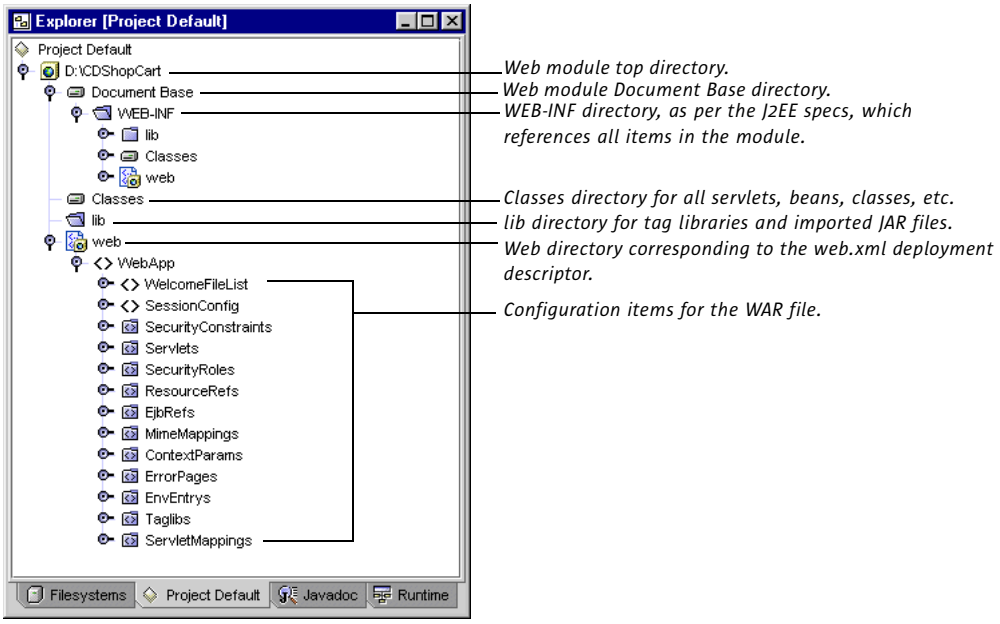
The New Web Module dialog box (see Step 3) is displayed, showing the `CDShopCart` directory.

10 Click Finish.

The new web module is created in the Explorer. You are prompted that an alternate view of the web module is installed in the Default Project tab window. Click OK to dismiss this message.

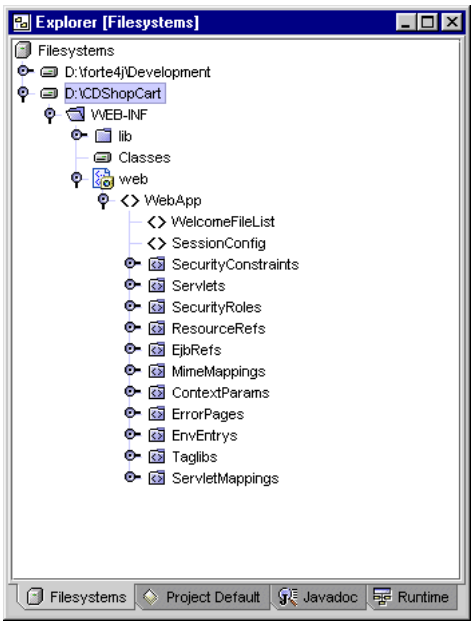
11 Click the Project tab in the Explorer to view the web module in the Project pane.

12 Open the icons in the web module to reveal what has been created automatically.




A somewhat truncated view of the web module is displayed in the Filesystems pane.

**13** Click the Filesystems tab in the Explorer to compare the view of the web module in that pane.




Throughout this tutorial, you will work mainly in the Filesystems pane.

Now you are ready to create the first component of the application, the `ProductList` JSP page.

# Using Forte for Java Custom Tags

In this section, you create the `ProductList` JSP page that fetches and displays the CD product data. In this tutorial, you use the Forte for Java custom database tags and their tag libraries to perform the database functions and the custom presentation tags to iterate through the fetched data.

## What is a JSP Tag?

A tag used in JSP pages is only one type of code that is allowed in the body of a JSP file. Here is a thumbnail description of the allowed types of code and how tags fit in.

The body of a JSP file can contain two types of code: fixed template data and elements.

- Fixed template data

  This is code of a type not known to the JSP container, and passes through the HTTP response unchanged. Examples of fixed template data are XML and HTML code, which you will use in the CDShopCart to create common HTML elements, such as headings, titles, tables, and buttons.

- Element types
  - Directives

    Used to declare global information about the JSP, such as which packages to import and whether the JSP must join a session.

  - Scripting elements

    Allow you to embed Java code within a JSP file.

  - Action elements

    XML-style tags that allow you to work with Java objects without having to write Java code. For example, you can use actions to locate and instantiate Java objects and get or set the object's properties.

## Tags (Action Elements)

There are standard action elements that are available in any JSP container. These elements are defined in the JSP specification document. You can also create custom action elements. Custom action elements are defined in an XML document called a *tag library*, which is made available to an individual JSP page by a declaration in a directive element. The tags you will learn to use for database and presentation in the CDShopCart application are custom tags that are provided in the Forte for Java IDE.

## Forte for Java Tag Libraries

Forte for Java provides three custom tag libraries that work together to create a visual presentation of row-based dynamic data, and can be used with many different kinds of data sources, such as JDBC ResultSets, EJBs, JDOs, and vectors or other collections of JavaBeans. These tag libraries are:

- ietags.jar (presentation and conditional tags)
- dbtags.jar (database tags)
- tptags.jar (transparent persistence tags)

The tag library source files are provided in the IDE. They are found in the `jsptaglibs_src.jar` file in the *`forte4j_home`*`/sources` directory.

**Where to get more information** The Forte for Java tag libraries are described in online help, in the Forte for Java Tag Libraries folder under the JavaServer Pages and Servlets main folder. You can find full descriptions of syntax, as well as short examples. A larger demo example is provided in the *`forte4j_home`*`/Development/Examples/TagLibDemo` directory. The `ReadMe` file in this directory tells you how to run the demo.

Tag libraries in general, including how you can make your own custom tags, are described in *Building Web Components*.

In the following sections, you will learn how to use database and presentation tags.

# Create the CD Catalog List Page

This section describes how to create the mechanism for retrieving data from the database you installed in Chapter 1, "Getting Started," and displaying it in a table for the user. The page you create looks like this:

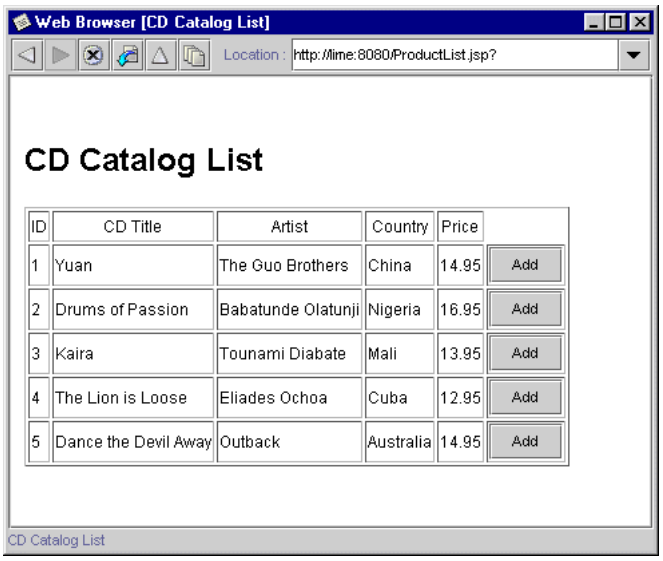



**Figure 2** *CD Catalog List Page*

To create this page, perform the following tasks:

1. "Add Forte for Java Tag Libraries to the Web Module" (description follows)
2. "Create the ProductList JSP Page" on page 46
3. "Declare the Tag Libraries" on page 47
4. "Use the JDBC connection Tag to Connect to the Database" on page 47
5. "Use the JDBC Query Tag to Fetch the CD Data" on page 48
6. "Iterate Through the Data With the Presentation Field Tag" on page 49
7. "Create the Add Button for Each CD Row" on page 50
8. "Clean Up With the JDBC cleanup Tag" on page 51

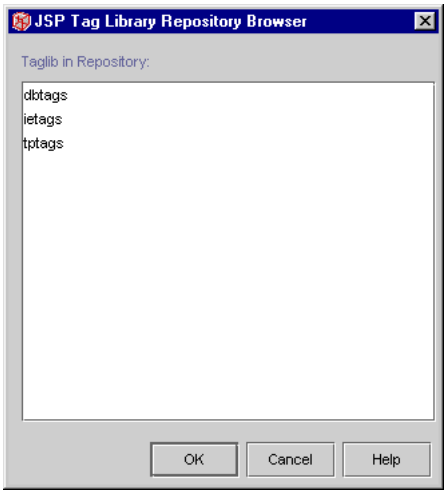
## Add Forte for Java Tag Libraries to the Web Module

In this section, you import two of the Forte for Java tag libraries—dbtags.jar (to implement database actions) and ietags.jar (to implement presentation actions)—to the `CDShopCart` web module, because you will use actions implemented by their tags.

➤ **To import Forte for Java tag libraries into the web module:**

1. In the Explorer, select the `CDShopCart` web module and choose Tools > Add JSP TagLibrary > Find in Tag Library Repository.

   The JSP Tag Library Repository Browser appears.

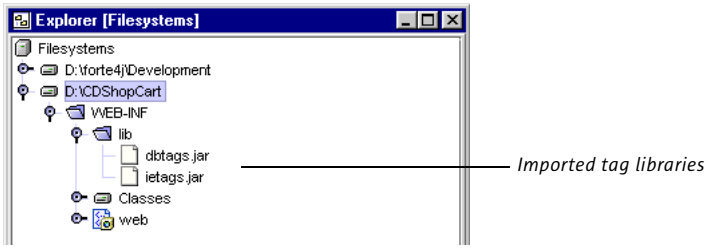



2. Select the dbtags and ietags libraries (use Shift or Ctrl when clicking), and click OK to add the libraries to the web module.

   In the Explorer, the `lib` folder under the `WEB-INF` folder opens.

3. Check that both tag libraries are there.

   The Explorer should look like this:




4. From the toolbar, choose Project > Settings and open the Filesystem Settings entry to see all the files that are now mounted in your classpath, including the tag libraries.

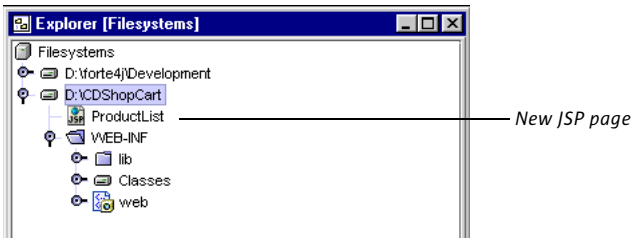
## Create the ProductList JSP Page

Now you are ready to create the page that uses the tags to retrieve the CD data from the database and display it in a table. The title of this page is the "CD Catalog List," and the mechanism that produces it is the `ProductList` JSP page.

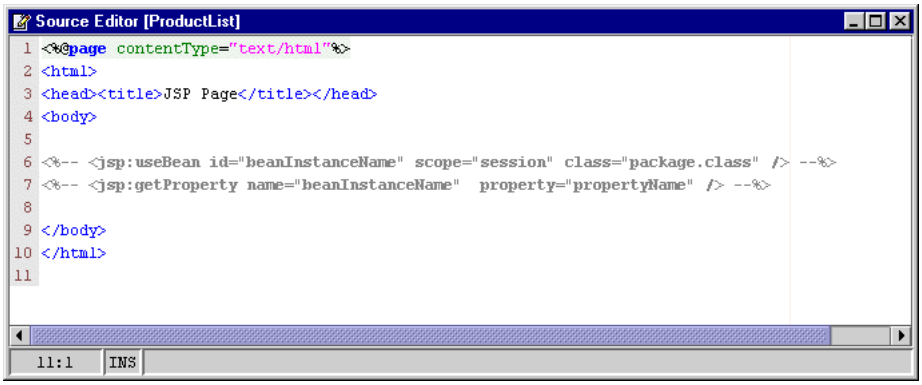
➤ **To create the ProductList JSP page:**

1. In the Explorer, right-click the `CDShopCart` web module and choose New > JSP & Servlet > JSP (HTML).
2. Type **`ProductList`** in the Name field and click Finish.

   The `ProductList` JSP page is displayed in the web module.




And a JSP page skeleton is displayed in the editor.




Note how the JSP editor features color coding and code completion.

## Declare the Tag Libraries

To use tags in a JSP, you must first declare the tag library with a `taglib` directive.

The `taglib` declaration declares that the page uses the tag library of a given URI, and specifies the tag prefix that is used in calls to actions in the library. The URI use for both tag libraries is their location in the web module (`/WEB-INF/lib`). The prefix is `jdbc` for the tags in `dbtags.jar` and `pr` for the presentation tags in `ietags.jar`.

You must put the directive above the body of the JSP, right under the page title. The following procedure shows how to change the title of the page, and then add the directives for the two tag libraries.

➤ **To declare the tag libraries in the ProductList JSP page:**

Change the page title to "CD Catalog List" and add the following directives to import the dbtags.jar and ietags.jar packages.

```
<head><title>CD Catalog List</title></head>
<%@taglib uri="/WEB-INF/lib/dbtags.jar" prefix="jdbc" %>
<%@taglib uri="/WEB-INF/lib/ietags.jar" prefix="pr" %>
```

Database tag lib directive

Presentation tag lib directive

JSP tags are based on XML syntax, and have one of two forms:

- Start tag (the element name) plus possible attribute/attribute value pairs, an optional body, and a matching end tag
- Empty tag with possible attributes

In this tutorial, you use both types of syntax. The first tag you will use is the JDBC `connection` tag, which is the empty tag type (with lots of attributes). This tag makes the connection to the database. (See the online help description for these attributes.)

## Use the JDBC connection Tag to Connect to the Database

The first JDBC tag you use is the `connection` tag. This tag creates a JDBC connection to a database. You have the option of storing the connection in any of four scopes: application, session, request, or page. The default scope is application. In this JSP, you use the default scope, because you want the connection to be global, for the whole application.

The `connection` tag has many attributes (see the online help description), but the attributes you will use are:

jdbc:connection tag

```
<jdbc:connection id="<connection_id>"
    driver="<driver_string>"
    url="<driver_url>"
    user="<user_id>" password="<pwd>" />
```

Put the `connection` tag below the body HTML tag. First, however, add a page title for the page, and then add the connection specification.

➤ **To use the JDBC connection tag:**

Below the BODY tag, create a header and the JDBC connection.

- The following is for a PointBase driver; read further for Oracle or SQLServer driver specifications.

Page title
Create a JDBC connection.
Use the PointBase database.

```
<body>
<h1> CD Catalog List </h1>
<jdbc:connection id="jdbcConn"
    driver="com.pointbase.jdbc.jdbcUniversalDriver"
    url="jdbc:pointbase://embedded/cdshopcart"
    user="PUBLIC" password="PUBLIC" />
```

- If you are using an Oracle database, use this:

Oracle JDBC specification (thin driver)

```
    driver="oracle.jdbc.driver.OracleDriver"
    url="jdbc:oracle:thin:@hostname:port#:SID"
    user="userid" password="password"
```

The default Oracle port number is 1521.

- If you are using a Microsoft SQLServer database with a Weblogic driver, use this:

Microsoft SQLServer using a Weblogic driver specification

```
    driver="weblogic.jdbc.mssqlserver4.Driver"
    url="jdbc:weblogic:mssqlserver4:database@hostname:port#"
    user="userid" password="password"
```

The default port number for SQLServer is 1433.

## Use the JDBC Query Tag to Fetch the CD Data

Now you will use another JDBC tag, `query`, to query the database. The `query` tag queries a database and gets the results. After executing the query, the generated ResultSet is stored against the resultsID in a pageContext. The ID can be passed to iterator tags to display the results. The `query` tag supports the standard SQL statement Select. Because the SQL statement is specified in the body instead of as an attribute, you can use JSP scripting to control how the query is created.

The `query` tag has the more complex tag syntax:

```
<jdbc:query id="<stored_query_id>" connection="<connection_id>"
resultsId="<results_id>" resultsScope="<scope>" >
    body
</jdbc:query>
```

You will put the `query` tag just following the `connection` tag.

➤ **To query the database for all the CD data:**

Create a query to select all the CD data from the database.

Use the "jdbcConn" connection created above, and put it on the session.

The SQL select statement:

End of query tag:

```
<jdbc:query id="productQuery" connection="jdbcConn"
resultsId="productDS" resultsScope="session" >

    SELECT * FROM CD
</jdbc:query>
```

## Iterate Through the Data With the Presentation Field Tag

At this point, you need to create a table, and then fill the table cells with the data. You will use two presentation tags, `iterator` and `field`, to iterate through the data you just fetched.

- `iterator`—iterates over rows in a data source Results. It understands the `results` interface and several specialized datasource types, including `JDBCResultSet`, `java.util.Vector,` and `java.util.Collection.`

  The syntax you will use is:

  ```
  <pr:iterator results="<results_id>" >
      body
  </pr:iterator>
  ```

- `field`—gets the value or the name of a field in the current row of results. Results are usually obtained from an enclosing row or field iterator, but they can also be specified directly. In this tutorial, you use the row iterator.

  The field that is displayed is determined by the `name` or `index` attributes. If the `name` attribute is present, then the tag retrieves the named column from the results. If the `index` attribute is present, then the tag retrieves the indexed column from the results. If neither is present, then the field tag must be enclosed in a `fieldIterator` tag, in which case it retrieves the current column of the current row of the `fieldIterator's` results.

  In this tutorial, you use the `name` attribute. The syntax is:

  ```
  <pr:field name="<field_name>"/>
  ```

In the next procedure, first create a table with HTML tags. Then, use the `iterator` tag with the results you specified in the `query` tag, above. Finally, use the `field` tag within the HTML syntax to place the specific column from the results.

➤ **To use the presentation tags to iterate through the data:**

1 Start a table for the CD data.

```
<TABLE border=1>
  <TR>
    <TH>ID</TH>
    <TH>CD Title</TH>
    <TH>Artist</TH>
    <TH>Country</TH>
    <TH>Price</TH>
  </TR>
```

Create headings for all the table columns.

2 Next, use the iterator and field tags to populate the table.

```
<pr:iterator results="productDS" >

  <TR>
    <TD><pr:field name="id"/></TD>
    <TD><pr:field name="cdtitle"/></TD>
    <TD><pr:field name="artist"/></TD>
    <TD><pr:field name="country"/></TD>
    <TD><pr:field name="price"/></TD>

</pr:iterator>
```

Start iterating the query results.

Retrieve the value from each field in the current row of the results.

End iterator tag:

## Create the Add Button for Each CD Row

Each row of the CD table holds the data for a CD. To purchase a CD, the user clicks the Add button on each row. You will create an HTML form to define the area for user input (clicking the button), and within this area, you will embed information that will be passed to the Shopping Cart JSP page. You put this code above the end `iterator` tag (you created in the last line, above).

➤ **To create the Add button:**

1 Create a form for the table within a cell (start above the `</pr:iterator>` tag you just created).

```
<TD>
<form method=get action="ShopCart.jsp">
```

Start form.

2 Specify the embedded information.

Add product ID.
Add product title.
Add product price.
Add **Add** button.

```
<input type=hidden name=cdId value="<pr:field name="id"/>">
<input type=hidden name=cdTitle value="<pr:field
name="cdtitle"/>">
<input type=hidden name=cdPrice value="<pr:field name="price"/>">
<input type=submit name=operation value=Add>
```

3 End the form, the cell, and the row.

```
</form>
</TD>
    </TR>
```

4 Below the iterator end tag, end the table.

Add this line:

```
</pr:iterator>
</TABLE>
```

## Clean Up With the JDBC cleanup Tag

The final code you need to add is a `cleanup` tag. The JDBC `cleanup` tag frees the resources being used by other tags on the JSP page. Its syntax is:

```
<jdbc:cleanup scope="<scope>" status="ok|error" />
```

See the online help for a fuller description of this tag.

After the cleanup tag, you need only end the body of the JSP.

➤ **To perform a cleanup of resources and end the page:**

1 Type the following code:

```
<jdbc:cleanup scope="session" status="ok" />
</body>
</html>
```

2 Choose File > Save to save your work.

## Test Run the ProductList JSP Page

Now you must test your typing. Compile the ProductList page, then use the integrated Forte for Java runtime system and browser to execute the page.

➤ **To test run the ProductList JSP page:**

1 In the Explorer, select the `CDShopCart` web module and choose Build > Build All.

Watch the message area in the lower part of the Toolbar for status messages. If all is well, you see "Finished." If there are problems, the output window displays an error message, with the problem line. Fix any problems and redo this step until you see the "Finished ..." message.

2 Test run the ProductList JSP by selecting it and clicking the Execute button in the Toolbar.

Or choose Build > Execute, or right-click ProductList and choose Execute from the contextual menu.

Forte for Java switches to the Runtime workspace and opens the Execution window. When the Servlet is running, a message is displayed in the Execution window and the browser opens. After a few seconds, the CD Catalog List page displays, as in Figure 2 on page 44.

3 Terminate the execution by right-clicking on the process in the Execution window and choosing Terminate Process.

If you want to re-execute ProductList, note that when you execute the program after the first time in a session, you must choose Execute (restart server).

4 Return to the Editing workspace, by clicking its tab.

Congratulations! You have successfully created a JSP page, and used Forte for Java custom tags to open a connection to a database, and retrieve and display data from it. Now you are ready to create the Shopping Cart page.

# Creating the Shopping Cart Page and Supporting Elements

In this section, you create the mechanism for displaying items selected for purchase from the CD Catalog List page. You create a bean (`CartLineItem`) to hold the attributes of the selected CD row passed as parameters from the ProductList page, and another bean (`Cart`) to hold the `CartLineItem` objects. You then create the `ShopCart` JSP page to receive the `Cart` objects and display them as a row in a table. Create and implement a Delete button for each item displayed on the `ShopCart` page. Finally, you place Cancel Order, Resume Shopping, and Place Order buttons on the page, with their implementations.

To create the `ShopCart` page and its beans, perform the following tasks:

1. "Create the CartLineItem JavaBeans Component," which follows
2. "Create the Cart JavaBeans Component" on page 57
3. "Create the ShopCart JSP Page" on page 59
4. "Test Run the Shopping Cart Page" on page 63

The page you create looks like this when a few items have been selected.

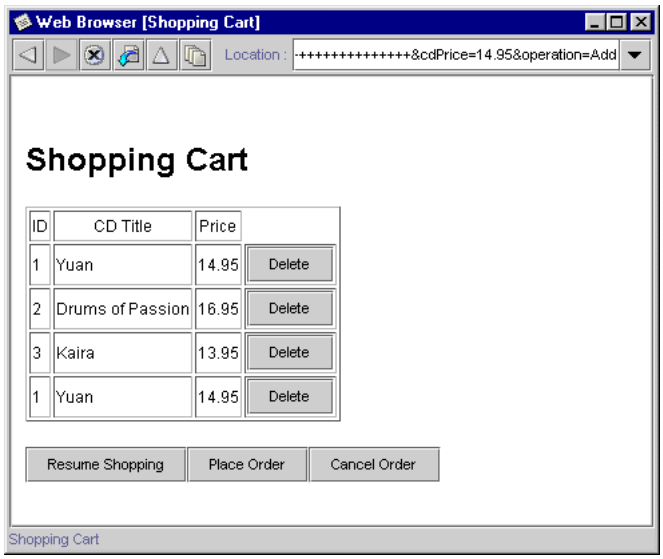



**Figure 3** *Shopping Cart Page*

# Create the CartLineItem JavaBeans Component

In this section, you create a line item bean whose object can hold the parameters passed to the Shopping Cart page from the CD Catalog List (`ProductList`) page. To do this, you will create three properties on the bean with their accessor methods.

➤ **To create the CartLineITem JavaBeans component:**

1 Open the `WEB-INF` folder of the `CDShopCart` web module, right-click the `Classes` folder, and choose New > Beans > Bean.

Use the GUI to create the bean.

2 In the New From Template New Object Name dialog box that appears, type **`CartLineItem`** and click Finish.

3 A message is displayed prompting you to add the new bean to the project; click Yes.

The new `CartLineItem` bean is displayed in the Explorer, and its code in the Editor.

4 Open the bean and its class to reveal its contents.

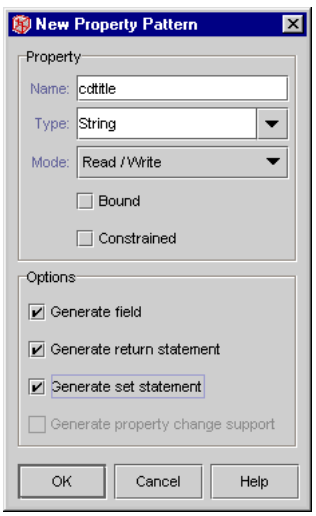
Create the cdtitle property.

5 Right-click on Bean Patterns and choose New > Property.

6 In the New Property Patterns dialog box, enter the following information:

Name: **cdtitle**
Type: **java.lang.String**
Generate field: **checked**
Generate return statement: **checked**
Generate set statement: **checked**

The dialog box should look like this:




7 Click OK to accept the information and close the dialog box.

Create the id property.

8 Similarly, create the id property with the following values:

Name: **id**
Type: **int**
Generate field: **checked**
Generate return statement: **checked**
Generate set statement: **checked**

Create the price property.

9 Create the price property with the following values:

Name: **price**
Type: **double**
Generate field: **checked**
Generate return statement: **checked**
Generate set statement: **checked**

10 Open the `Fields` folder (of the `CartLineItem` bean class) to see the new fields you created.

11 Open the `Methods` folder to see the new `get` and `set` methods for each field.

12 Double-click on a new field (or a new method) to see the new code that was created in the Editor.

The parameters passed from the `ProductList` JSP page are all passed as strings. However, because neither the `id` or `price` properties are strings, you will have to convert them. An efficient way to do this is to overload the properties' setter methods and add the proper code.

➤ **To overload the setId and setPrice methods:**

Overload the setId method to convert id to a string.

1 Right-click on the `Methods` folder (of the `CartLineItem` bean), and choose New Method...

2 In the Edit new method dialog box that appears, enter the following:

Name: **setId**
Return Type: **void**

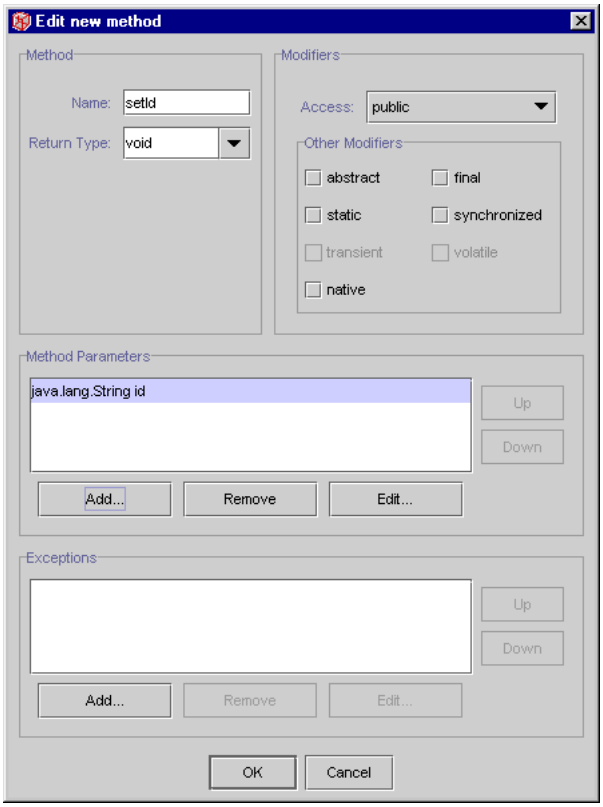
3 In the Method Parameters box, click the **Add** button to display the Enter Method Parameter dialog box.

4 Enter the following values:

Type: **java.lang.String**
Name: **id**

5 Click OK.

The New Method dialog box should look like this:




6 Click OK to create the method and close the dialog box.

7 Add code to this new method:

Add this line:
And this line:

```
public void setId(java.lang.String pId) {
  int val = Integer.parseInt(pId);
  this.setId(val);
}
```

Now do the same with the `setPrice` method.

Create a new setPrice method.

8 Create a new `setPrice` method with these values:

Method –
Name: **setPrice**
Return Type: **void**

Method Parameter –
Type: **java.lang.String**
Name: **price**

9 Add the following code to this new method:

Add this line:
And this line:

```
public void setPrice(java.lang.String pPrice) {
  double val = Double.parseDouble(pPrice);
  this.setPrice(val);
}
```

Compile the bean.

10 Select the `CartLineItem` bean (not the class) and click the Compile button.

If the bean compiles without errors, you are ready to create the `Cart` bean. If not, check your typing and recompile.

You are now ready to create the `Cart` bean.

## Create the Cart JavaBeans Component

The Shopping Cart JSP page instantiates (or finds, if it already exists) a cart object to hold the CD line item objects that are passed to it by the `ProductList` JSP page when a user clicks the **Add** button. The `cart` object is based on a `Cart` JavaBeans component.

➤ **To create the Cart bean:**

1 Right-click the `Classes` folder under the `WEB-INF` folder of the web module and choose New > Beans > Bean.

2 Name the bean **Cart**.

Create the lineItems property.

3 Right-click on Bean Patterns and choose New > Property.

4 In the New Property Patterns dialog box, enter the following values:

Name: **lineItems**
Type: **java.util.Vector**
Generate field: **checked**
Generate return statement: **checked**
Generate set statement: **checked**

5 Double-click on the new `lineItems` field (or the new methods) to see the new code that was created in the Editor.

You need to change the access of the field from "private" to "public."

6 Open the `Fields` folder of the `Cart` class and select the `lineItems` field.

7 Open its Properties window and change the Modifiers value to **public**.

Now add code that instantiates a line item object, and a method that returns the element number of a selected item and another method that removes a line item from the cart.

➤ **To add the required code:**

1 Add code to the `Cart` bean's constructor to instantiate a new lineItems object, as follows:

Add this line:

```
    public Cart() {
        propertySupport = new PropertyChangeSupport ( this );
        lineItems = new java.util.Vector();
    }
```

Create the findLineItem method.

2 Right-click on the Cart `Methods` folder, choose New Method..., and enter the following values for this method:

Method –
Name: **findLineItem**
Return Type: **int**

Method Parameter –
Type: **int**
Name: **pID**

3 Add the following code in the Editor to the `findLineItem` method:

Start here:
Returns the element number of the item in the cartItems as specified by the passed ID.

```
public int findLineItem(int pID) {
  System.out.println("Entering Cart.findLineItem()");
  int cartSize = (lineItems == null) ? 0 : lineItems.size();
  int i ;
  for ( i = 0 ; i < cartSize ; i++ )
  {
    if  ( pID == ((CartLineItem)lineItems.elementAt(i)).getId() )
        break ;
  }
  if (i >= cartSize) {
    System.out.println("Couldn't find line item for ID: " + pID);
    return -1 ;
  }
  else
    return i ;
}
```

End here:

4 Create the `removeLineItem` method and enter the following values:

Method –
Name: **removeLineItem**
Return Type: **void**

Method Parameter –
Type: **int**
Name: **pID**

5 Add the following code in the Editor to the `removeLineItem` method:

Start here:
Removes a cartItem from the cartItems list.
End here:

```
public void removeLineItem(int pID) {
  System.out.println("Entering cart.removeLineItem()");
  int i = findLineItem(pID);
  if (i != -1) lineItems.remove(i);
  System.out.println("Leaving cart.removeLineItem()");
}
```

Compile the Cart bean.

6 Select the `Cart` bean (not the class) and click the Compile button to compile the `Cart` bean.

The bean should compile without errors.

You are ready to create the `ShopCart` JSP page.

## Create the ShopCart JSP Page

Now you are ready to create the page that receives the parameters passed from the CD Catalog List page and displays some of them (id, title, and price) as a row in a table. This page also offers mechanisms for deleting an item from the table, returning to the Catalog List page, and placing the order. The title of this page is "Shopping Cart," and the mechanism that produces it is the `ShopCart` JSP page.

➤ **To create the ShopCart JSP page:**

1 Create a JSP page by right-clicking the `CDShopCart` web module and choosing New > JSP & Servlet > JSP (HTML).

2 Name the JSP page **`ShopCart`** and click Finish.

The `ShopCart` JSP is displayed in the Explorer and in the Editor.

To develop this page, perform the following tasks:

1 "Add Code to Add or Remove an Item From the Shopping Cart Table," which follows

2 "Use Presentation Tags to Populate the Cart Table" on page 61

3 "Add the Buttons to the Page" on page 62

## Add Code to Add or Remove an Item From the Shopping Cart Table

In this section, you add code that creates the cart items table. You will instantiate a `Cart` object and a `CartLineItem` object, so you must use a directive to import the `Cart` bean and the `java.util` library (the `CartLineItem` is a type Vector, from this library). You will use the same presentation tags that you used in the `ProductList` JSP page, so you must also add a directive to import these tags.

You use a scriptlet to create the cart and either add a line item created with the parameters passed from the `ProductList` JSP page, or delete a line item (when you press the **Delete** button). If the table is empty, you will forward to a JSP page you haven't created yet (the `EmptyCart` JSP page).

As with the `ProductList` JSP page, you use iteration tags to organize the table data, and then use a form to create the table and add a Delete button to each row.

➤ **To code the ShopCart JSP page:**

1 Add a Page directive to import the `java.util` library and the `Cart` bean.

Add this line:

```
<%@ page contentType="text/html" %>
<%@ page import="java.util.*, Cart" %>
```

2 Change the page title to "Shopping Cart" and add the directive to import the and `ietags.jar` library.

Change the title.

Presentation tag lib reference.

```
<head><title>Shopping Cart</title></head>
<%@taglib uri="/WEB-INF/lib/ietags.jar" prefix="pr" %>
```

3 Below the BODY tag, create a "Shopping Cart" header for the page.

Add this line:

```
<body>
<h1> Shopping Cart </h1>
```

4 Below the header, use the `usebean` tag to tell the JSP to use the Cart bean.

Instantiate a Cart object and place it on the session.

```
<jsp:useBean id="myCart" scope="session" class="Cart" />
```

Now you must specify what happens when the current operation for the session is "Add." This happens when the user clicks the Add button on the `ProductList` page. The code gets the `cdID`, `cdTitle`, and `cdPrice` objects and adds them to the `myCart` object.

5 Begin creating the scriptlet with the following code:

To add an item to the Cart:

```
<%
  String myOperation = request.getParameter("operation");
    session.setAttribute("myLineItems", myCart.getLineItems());

  if (myOperation.equals("Add"))
    {
    CartLineItem lineItem = new CartLineItem();
    lineItem.setId(request.getParameter("cdId"));
    lineItem.setCdtitle(request.getParameter("cdTitle"));
    lineItem.setPrice(request.getParameter("cdPrice"));
    myCart.lineItems.addElement(lineItem);
    }
```

Now, specify what happens when the current operation for the session is "Delete." This happens when the user clicks the Delete button on the `ShopCart` page.

**6** Type the following code:

To remove an item:

```
if (myOperation.equals("Delete"))
  {
  String s = request.getParameter("cdId");
  System.out.println(s);
  int idVal = Integer.parseInt(s)
  myCart.removeLineItem(idVal);
  }
```

Finally, you must specify what happens when the Delete action deletes the last row of the Cart CD table. Use the JSP `forward` tag to go to the `EmptyCart` HTML page (which you will soon create). This last code ends the script you started in Step 5 (though you have to make a break for the `forward` tag, then resume before you finally end the scriptlet).

**7** Type the following code:

When the last item is removed from the Cart, (end scriptlet temporarily) forward to the EmptyCart page (begin scriptlet again).

End script.

```
if (((Vector)session.getAttribute("myLineItems")).size() == 0)
  {
  %>
  <jsp:forward page="EmptyCart.html" />
  <%
  }
  %>
```

## Use Presentation Tags to Populate the Cart Table

Next, you will use the iterator and field tags to iterate through the passed data, much as you did in "Iterate Through the Data With the Presentation Field Tag" on page 49.

➤ **To use the presentation tags to iterate through the data:**

**1** Start a table for the purchase candidate data.

Create the table headings.

```
<TABLE border=1>
  <TR>
    <TH>ID</TH>
    <TH>CD Title</TH>
    <TH>Price</TH>
  </TR>
```

**2** Use the `iterator` and `field` tags to populate the table:

Start the row iterator.

```
<pr:iterator results="myLineItems" >
  <TR>
```

Retrieve the value from each field in the current row. of the results.

```
    <TD><pr:field name="id"/></TD>
    <TD><pr:field name="cdtitle"/></TD>
    <TD><pr:field name="price"/></TD>
```

3 Create a Delete button for each row, as in"Create the Add Button for Each CD Row" on page 50.

```
<TD>
<form method=get action="ShopCart.jsp">
<input type=hidden name=cdId value="<pr:field name="id"/>">
<input type=hidden name=cdTitle value="<pr:field
name="cdtitle"/>">
<input type=hidden name=cdPrice value="<pr:field name="price"/>">
<input type=submit name=operation value=Delete>
</form>
</TD>
    </TR>
</pr:iterator>
</TABLE>
```

Start form.
Add product id.
Add product title.
Add product price.
Add Delete button.
End form.
End cell.
End row.
End iterator from line 44.
End table.

## Add the Buttons to the Page

Finally, add the Resume Shopping, Place Order, and Cancel Order buttons to the page bottom.

➤ **To add the buttons to the page:**

1 Add the following code to the ShopCart JSP page.

```
<p>
<form method=get action="ProductList.jsp">
<input type=submit value="Resume Shopping">
</form>
<form method=get action="PlaceOrder.jsp">
<input type=submit value="Place Order">
</form>
<form method=get action="CancelOrder.jsp">
<input type=submit value="Cancel Order">
</form>

</body>
</html>
```

Create the Resume Shopping button.
Create the Place Order button.
Create the Cancel Order button.
End the page.

## Test Run the Shopping Cart Page

You don't test run the Shopping Cart page directly. You test run the `ProductList` page and navigate (by means of the **Add** button) to the Shopping Cart page.

➤ **To test run the application so far:**

1. Select the `CDShopCart` web module and choose Build > Build All.

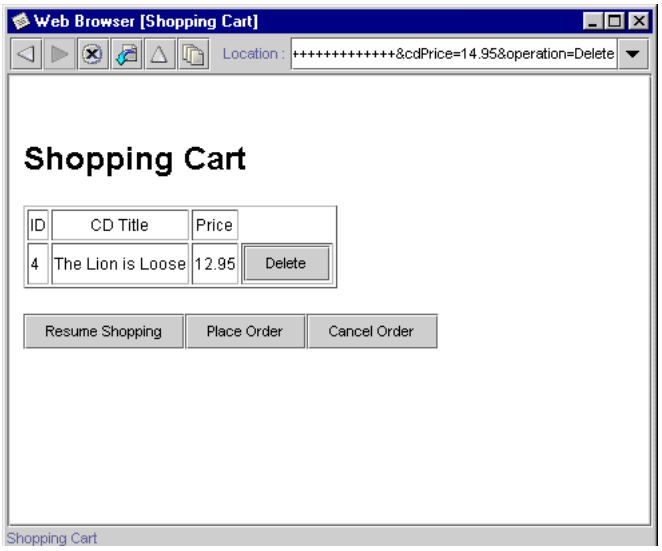
   Everything should compile successfully.

2. Right-click the `ProductList` JSP page and choose Execute (restart server).

   After a few seconds, the CD Catalog List page is displayed.

3. Click one of the Add buttons to navigate to the Shopping Cart page.

   The Shopping Cart page should appear something like this.




4. Use the Resume Shopping button to return to the CD Catalog List page.

Note Using the Back button on the ICE Browser produces unexpected results, because it does not cache the pages the way Netscape or Internet Explorer do. If you are using either Netscape or IE, you can use the Back button as well as the Resume Shopping button.

5. Terminate the execution by right-clicking on the process in the Execution window and choosing Terminate Process.

6. Return to the Editing workspace by clicking the Editing tab of the Explorer window.

Congratulations! You have almost finished the CDShopCart application. You only have to create the EmptyCart and PlaceOrder pages, and you're done!

# Creating the Three Message Pages

In this section, you create a JSP page that displays when a user empties the cart and two more that display when the user clicks the **Place Order** and **Cancel Order** buttons, respectively.

The three pages you create in this section are:

- "Empty Cart Page," which follows
- "Place Order JSP Page" on page 65
- "Cancel Order JSP Page" on page 67

## Empty Cart Page

When an iterator tag finds an empty vector, it throws an exception (rather than creating an empty table). We have dealt with this by testing for this case (lines 28 through 33 in "Add Code to Add or Remove an Item From the Shopping Cart Table" on page 59) and then handling it by displaying the Empty Cart page. This page contains a Resume Shopping button, to allow the user to return to the application.

Note For alternative ways to handle the empty vector situation, see the examples distributed with Forte for Java, Internet Edition. They are found in the `examples` folder in the `Development` file system.

The Empty Cart page looks like this.

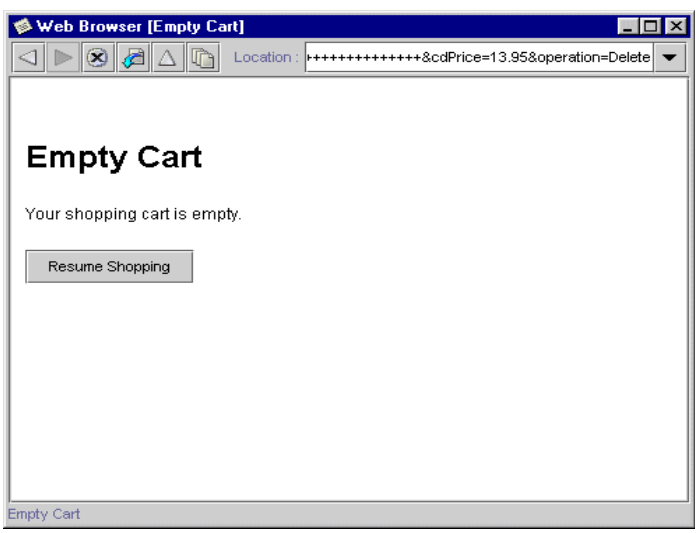



**Figure 4** *Empty Cart Page*

➤ **To create the Empty Cart page:**

1. Create an HTML page by right-clicking the `CDShopCart` web module and choosing New > Other > HTML File.
2. Name it **EmptyCart** and click Finish.

   The file is displayed in the browser.
3. Close the browser.
4. To view the source file in the Editor, right-click the EmptyCart HTML file and choose Open.
5. Add the following HTML code to the page:

```
<HTML>
<head><title>Empty Cart</title></head>
<body>
<h1> Empty Cart </h1>
Your shopping cart is empty.
<p>
<form method=get action="ProductList.jsp">
<input type=submit value="Resume Shopping">
</form>
</body>
</HTML>
```

Create a title.

Add a heading.

Add a message.

Add a space

Add a button to return to the Catalog page.

## Place Order JSP Page

This and the CancelOrder page are very simple pages, and present only one of many ways you can resolve these actions. Because programming these pages demonstrates little more of the Forte for Java features than you have already seen, we have chosen the simplest possible resolution.

The Place Order page displays when the user clicks the Place Order button on the Shopping Cart page. Displaying this page ends the session. The page looks like this:

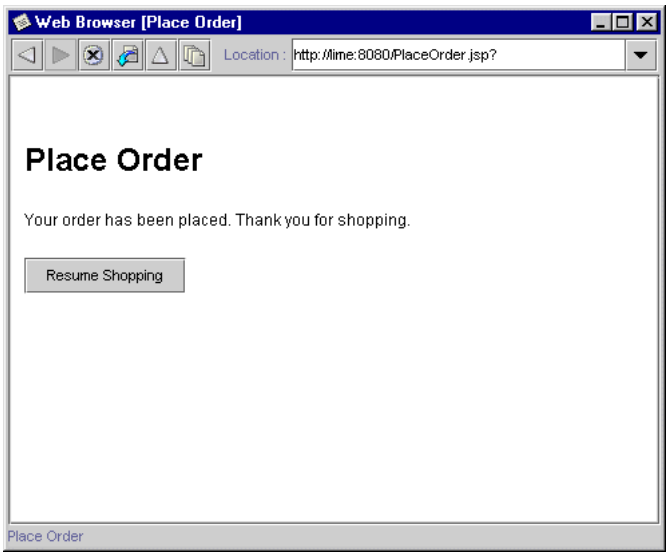



**Figure 5** *Place Order Page*

➤ **To create the Place Order page:**

1. Create a JSP page by right-clicking the `CDShopCart` web module and choosing New > JSP & Servlet > JSP (HTML).
2. Name it **PlaceOrder** and click Finish.
3. Change the page title to "Place Order" and add code as follows:

Change the title.

Add a heading.

Invalidate the session.

Enter the message.
Add a space.
Add a button to return to the Catalog page.

```
<%@page contentType="text/html"%>
<html>
<head><title>Place Order</title></head>
<body>
<h1> Place Order </h1>
<%
session.invalidate();
%>
Your order has been placed. Thank you for shopping.
<p>
<form method=get action="ProductList.jsp">
<input type=submit value="Resume Shopping">
</form>
</body>
</html>
```

## Cancel Order JSP Page

The Cancel Order page displays when the user clicks the Cancel Order button on the Shopping Cart page. Displaying this page ends the session. The page looks like this:

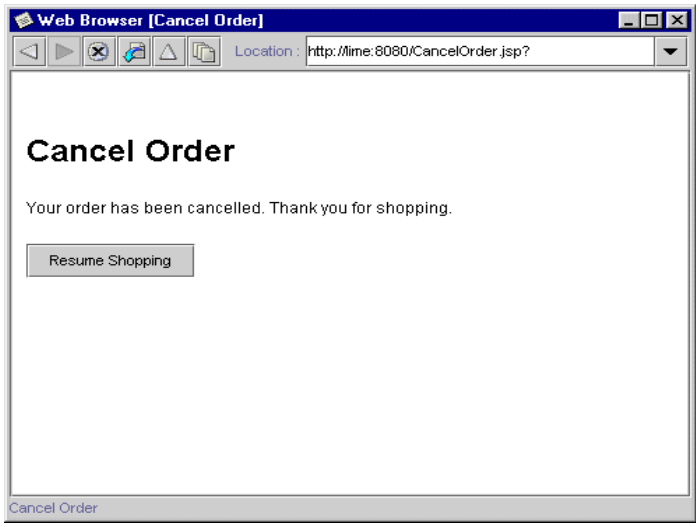



**Figure 6** *Cancel Order Page*

➤ **To create the Cancel Order page:**

1. Create a JSP page by right-clicking the `CDShopCart` web module and choosing New > JSP & Servlet > JSP (HTML).
2. Name it **CancelOrder** and click Finish.
3. Change the page title to "Cancel Order" and add code as follows:

Change the title.

Invalidate the session.

Enter the message.

Add a space.

Add a button to return to the Catalog page.

```
<%@page contentType="text/html"%>
<html>
<head><title>Cancel Order</title></head>

<body>
<h1> Cancel Order </h1>
<%
session.invalidate();
%>
Your order has been cancelled. Thank you for shopping.
<p>
<form method=get action="ProductList.jsp">
<input type=submit value="Resume Shopping">
```

```
</form>
</body>
</html>
```

## Test Run the Three Message Pages

As with the Shopping Cart page, you test run the message pages by running the ProductList page, adding CD items to the Shopping Cart, and then performing the appropriate action that displays each message page.

➤ **To test the message pages:**

1. Select the `CDShopCart` web module and choose Build > Build All.

   Everything should compile successfully.

2. Right-click the `ProductList` JSP page and choose Execute (restart server).

   After a few seconds, the CD Catalog List page is displayed.

3. Click one of the Add buttons to navigate to the Shopping Cart page.

4. To test the Empty Cart page, click the Delete button on the item you just put into the cart.

   The Empty Cart page should appear.

5. Click the Resume Shopping button to return to the CD Catalog List page.

6. Add one or more CDs to the cart.

7. Test the Cancel Order page by clicking the Cancel Order button.

8. When the page appears, click the Resume Shopping button to return to the Catalog page.

9. Add another CD to the cart.

10. When the Shopping Cart page appears, make sure that the CD you added is the only one in the cart.

    There should be only this CD in the cart because the Cancel Order ended the previous session.

11. Add more CDs to the cart, and then test the Place Order button.

12. When the Place Order page appears, click the Resume Shopping button to return to the Catalog page.

13. Add another CD to the cart.

    As with Step 10, because the Place Order page ended the session, only one CD should be in the cart.

14. To stop the application, right-click on the message in the Execution window and choose Terminate Process.

# Chapter 4

# Adding Transparent Persistence to the Tutorial Application

This chapter teaches you some basic techniques for using Transparent Persistence to interact with a database, by showing you how to use it to save a customer's order. You must already have created the basic CDShopCart application, as described in Chapter 3, "Creating the Basic Tutorial Application," before you can begin this chapter.

The topics covered in this chapter are:

# Overview of Transparent Persistence

Forte for Java Transparent Persistence is a preview of the Transparent Persistence technology described in the *Java Data Objects Specification*, which is available for public review at `http://java.sun.com/aboutJava/communityprocess/review/jsr012/index.html`. The book *Programming Persistence* provides details of what portion of this technology is offered in Forte for Java, Internet Edition.

The purpose of Transparent Persistence is to let you access information from a data store as Java objects, so that you can manipulate the data using the Java programming language, rather than SQL or some other data store-specific language. Transparent Persistence does this by mapping tables in a database schema to Java classes, and then enhancing them to be persistence-capable.

You use standard Java language operations and Transparent Persistence method calls on these persistence-capable classes to access the database. When you compile and run your application, the Transparent Persistence runtime system automatically performs and manages persistence operations.

You create persistence-capable classes from database schema tables by two methods: by automatically generating class definitions from specific tables within the schema, or by custom mapping existing classes to specific tables within the schema.

Persistence-capable classes can have both persistent fields and transient fields. *Persistent fields* represent persistent data and are managed by the Transparent Persistence runtime environment. The runtime environment synchronizes the field's value with the data store, flushes class values to the data store, and so on, in accordance with current transaction status and concurrency management strategy.

*Transient fields* are normal Java language constructs, managed by application logic, and don't participate in the Transparent Persistent mechanism. The application can use them for such things as values derived from persistent values, values used in a transaction that don't need to be saved to the data store, and so on.

**For more information** All aspects of Transparent Persistence are fully described in the book *Programming Persistence*. There is also online help on Transparent Persistence, in the Transparent Persistence folder and its subfolders. You can also browse Javadoc documentation of Transparent Persistent methods in the Javadoc pane of the Forte for Java Explorer.

## How You Use Transparent Persistence

Using Transparent Persistence in development of an application has two steps. In the first step, you create the persistence-capable classes from the database schema. In the second step, you use methods of Transparent Persistence classes and runtime environment objects to work with the data.

The basic sequence for calling Transparent Persistence methods is as follows:

1. Create or obtain a Persistence Manager Factory.

   The Persistence Manager Factory is the factory from which you create Persistence Managers to manage the database operations. The Persistence Manager Factory creates a database connection template for its Persistence Managers.

2. Create a Persistence Manager.

   A Persistence Manager Factory must exist before you can create a Persistence Manager. Each session usually creates its own Persistence Manager, which uses the database connection defined by the Persistence Manager Factory (although it can override this connection). Use the Persistence Manager is the factory for creating transaction objects and query objects. The query class provides a set of methods that serve the same purpose as database query functions.

3. (Optional) Create a Connection Factory.

   This is required only if you want to implement connection pooling. See the book *Programming Persistence* for more information.

4. Create an instance of the `Transaction` class.

   Use Persistence Manager methods to create the `Transaction` object, and use `Transaction` methods for all transaction operations on instances managed by the Persistence Manager.

5. Use the Transparent Persistence API to connect to the database and start and end transactions.

   All the business logic of your application (queries and updates to the database) is enclosed within the context of the transaction and the database on which the transaction has been started.

6. Invoke the business logic of your application.

   As the application queries the database, modifies records, and adds new records, it creates a set of persistent instances that represent the data it needs. The Persistence Manager manages all the database interactions for this set of instances.

7. Commit or abort the transaction.

   Commit the transaction to save your updates to the database. Abort the transaction to roll back the database to what it was before your transaction began. After commit, deleted objects become transient.

8. Perform additional transactions.

   You can use the same transaction object and execute additional logic, or repeat the logic that you have just executed. Or, you can create and use another transaction object.

9. Close the database and exit the application.

## Using Transparent Persistence in the CDShopCart Application

The CDShopCart application you created in Chapter 3, "Creating the Basic Tutorial Application" allows a user to create an order, but when the user clicks the Place Order button, it only displays a message about the order being placed. It doesn't actually save the order to the database. In this chapter, you add components that save the order.

The main steps in fulfilling the Place Order function are:

1. Create persistence-capable classes to generate and keep track of sequence numbers, assign them to each order item, then store the order and identifying numbers.

   In "Creating the Persistence-Capable Classes" on page 73, you use the tables you installed in the database in Chapter 1, "Getting Started."

2. Enhance the persistence-capable classes.

   Enhancing is the process by which Transparent Persistence automatically generates all of the necessary JDBC statements for the classes. You can enhance classes either by running them in the IDE or by packaging them. In "Enhance the Persistence-Capable Classes" on page 80, you package them in a JAR file within the `CDShopCart` web module.

3. Create a bean component to encapsulate all the Transparent Persistence technology for writing the order to the database.

   In "Creating the Persistence-Aware Bean" on page 82, you create the `CheckOutBean` bean that performs the following actions:

   a. Creates a Persistence Manager Factory.

   b. Creates a Persistence Manager.

      Usually, you create the Persistence Manager Factory at the application level, and the Persistence Manager at the session level. Creating both with the same component is a simplification required by this small application.

   c. Opens the database connection.

   d. Creates an instance of the `Transaction` class.

   e. Invokes the logic that assigns the identifying numbers to each order and its items.

   f. Commits the transaction, which saves the updates to the database.

   g. Returns the order number to the PlaceOrder JSP page.

4. Modify the PlaceOrder page to use `CheckOutBean` to place the order and get the order number it displays as part of the new message that the order has been placed.

   You perform this action in "Modifying the PlaceOrder Page to Call CheckOutBean" on page 89.

5. Test run the entire application.

   You do this in "Test Running the New CDShopCart Application" on page 90.

# Creating the Persistence-Capable Classes

Before mapping any Java classes to a database schema, you must capture the schema. This creates a working copy in your Filesystem that you can use without a live connection to the database. From this schema, you generate the persistence-capable classes from the three tables you created for the CDShopCart application. You then compile the package and enhance it.

In this section, perform the following procedures:

1. "Capture the Database Schema," which follows
2. "Generate the Persistence-Capable Classes" on page 76
3. "Enhance the Persistence-Capable Classes" on page 80

## Capture the Database Schema

The captured schema must be stored in a package. Later, you enhance the classes by packaging them in a JAR file, after which, you must unmount the original package so that it will not be in your CLASSPATH (all mounted directories are in your CLASSPATH). You do this because you want the application to use the enhanced classes (in the JAR file) rather than the unenhanced ones (in the package). Therefore, you must first mount a directory on your filesystem to hold the package.

You mount the directory, create the package in the directory, and then start the Database Schema wizard to capture the schema. You will not create the package in the `CDShopCart` web module, because only the JAR file that you package it in needs to be contained in the web module.

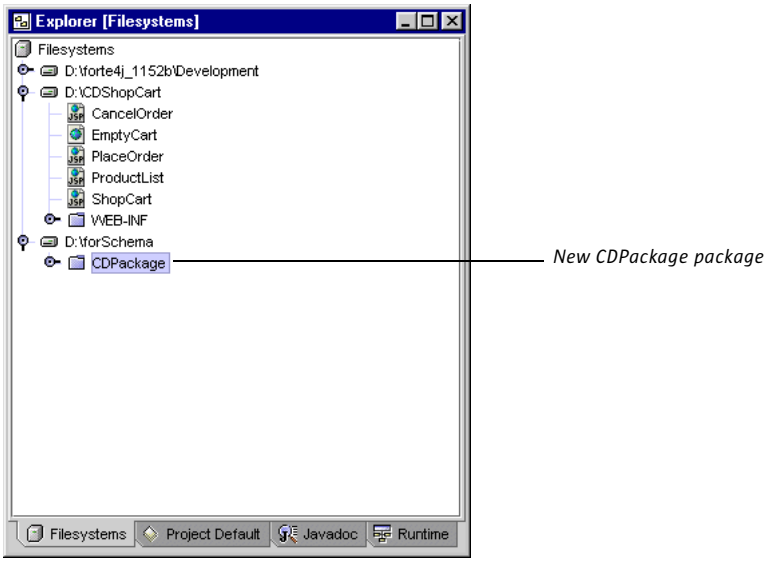
➤ **To capture the tutorial database schema:**

1. Create a directory somewhere on your file system.

   In this example, this directory is the **`forSchema`** directory.
2. Mount this directory by choosing File > Mount Filesystem.

   The `forSchema` folder appears in the Forte for Java Explorer.
3. Right-click the `forSchema` folder and choose New Package from the pop-up menu.
4. Name the package **`CDPackage`** and click OK.
5. When the prompt appears for you to add the new package to the current project, click Yes.

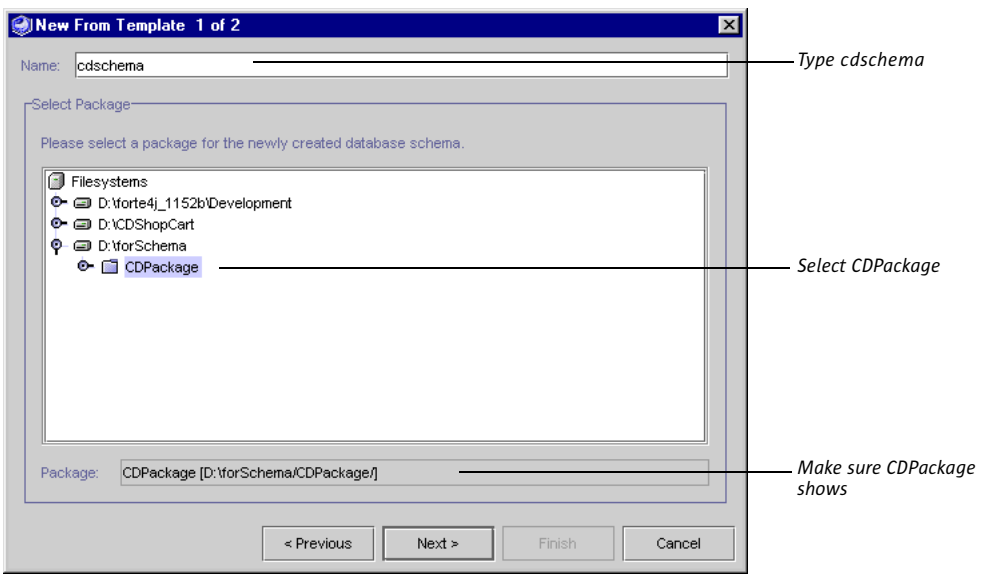
The new package appears in the Explorer.




6 Right-click on the `CDPackage` package and choose New > Databases > DB Schema.

The Database Schema wizard appears.

7 On the first page of the wizard, type **`cdschema`** for the name of the schema, find `CDPackage` in the Explorer pane, and select it, as shown.



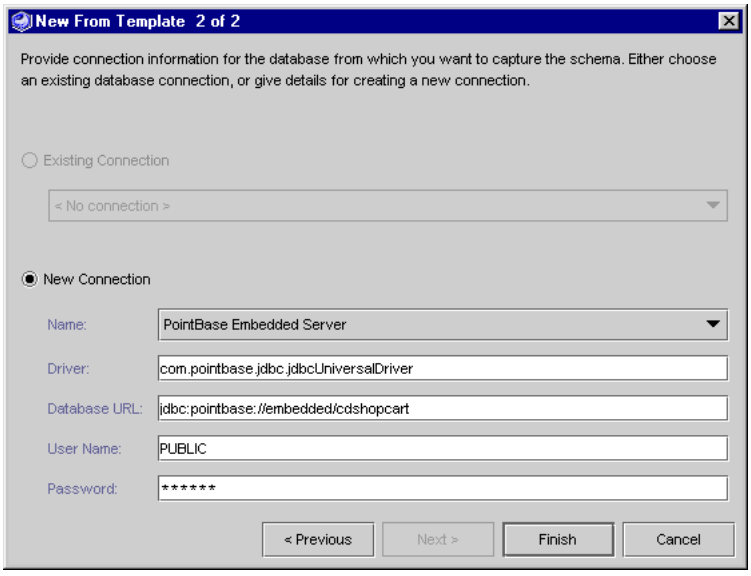

8 Click Next to go to the next page of the wizard.

9 On the second page, enter the following values:

New Connection: **enabled**
Name: **PointBase Embedded Server** (use the drop-down list)
Driver: **com.pointbase.jdbc.jdbcUniversalDriver**
Database URL: **jdbc:pointbase://embedded/cdshopcart**
User: **PUBLIC**
Password: **PUBLIC** (displayed as asterisks)

If you are using Oracle or Microsoft SQLServer, enter instead the same data that you used in "Use the JDBC connection Tag to Connect to the Database" on page 47.

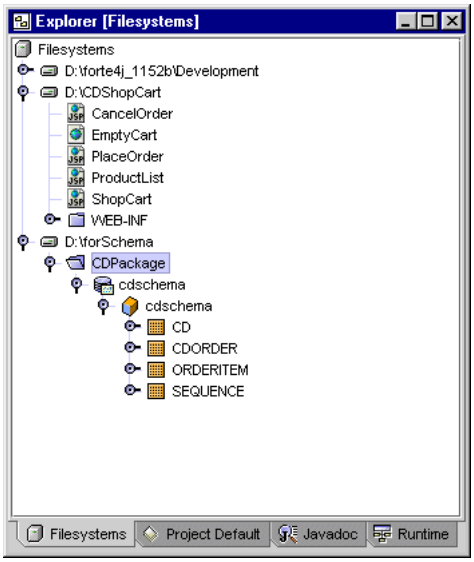
The wizard should look like this:




10 Click Finish.

A progress window appears. The new database schema appears in the `CDPackage` package.

**11** Open the package and the schema, to see the captured tables, as shown.




If you have more tables in your schema, they all will appear in the Explorer.

You are now ready to generate the persistence-capable classes.

## Generate the Persistence-Capable Classes

In this section, you generate the persistence-capable classes from some of the tables in the database schema that you just captured.

➤ **To generate persistence-capable classes:**

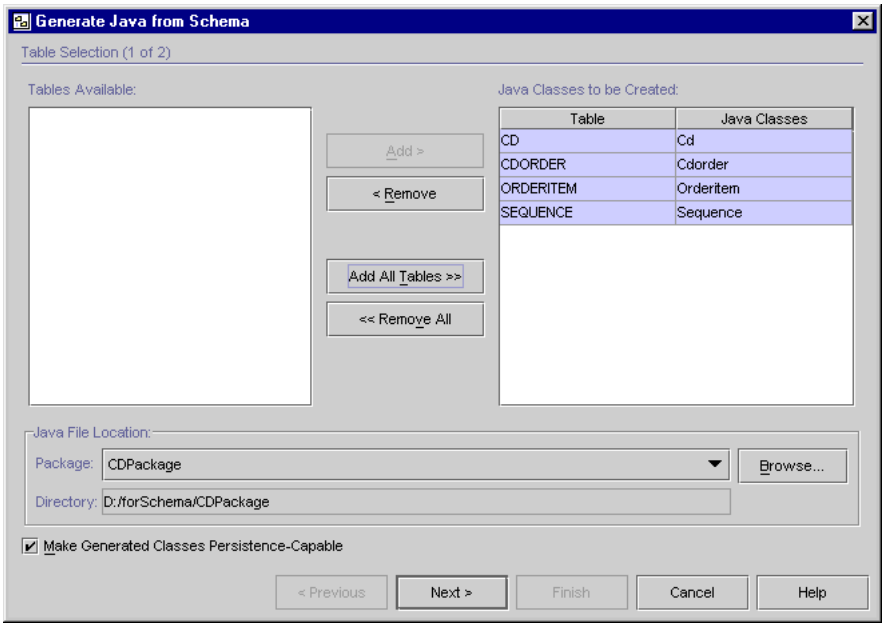
**1** Right-click the cdschema icon just under the `CDPackage` package, and choose Generate Java... from the pop-up menu.

This displays the Generate Java wizard.

**2** Click the Add All Tables to add the CD, CdOrder, OrderItem, and Sequence tables to the right pane of the dialog box.

Note If your schema has additional tables, use the Add button to add only these tables.

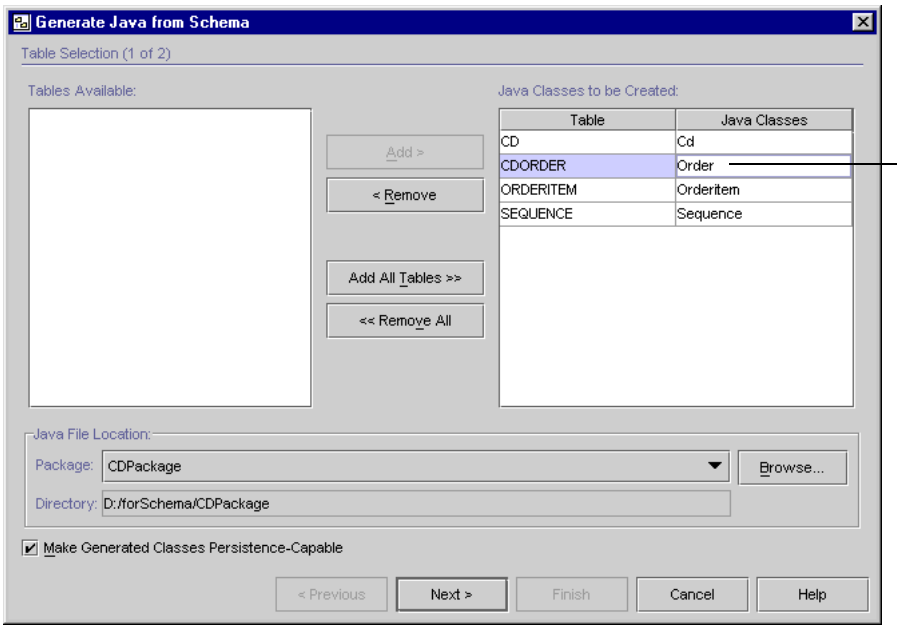
The Generate Java wizard window should look like this:




3 Click on `CdOrder` label under Java Classes, change the name to **`Order`**, and press Enter.

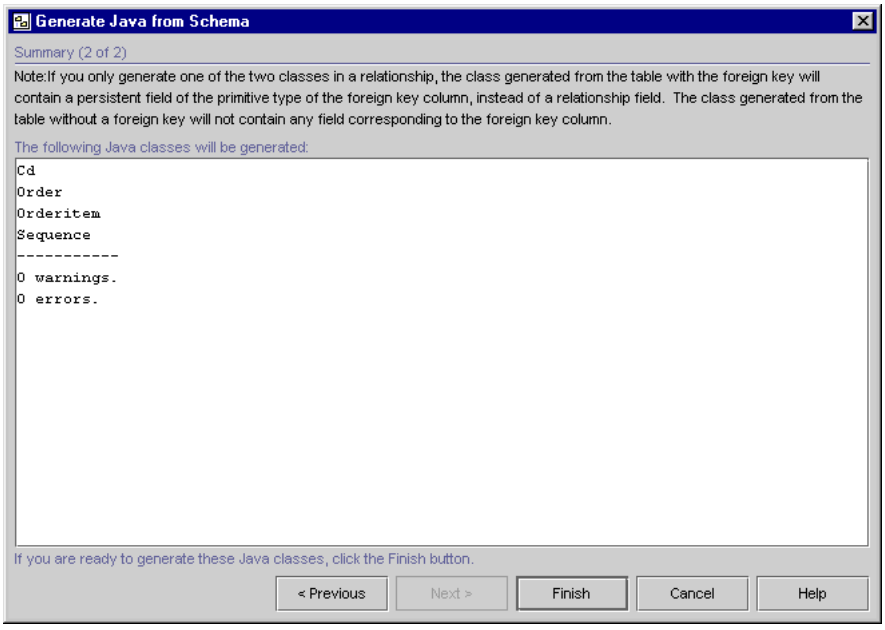
This exercise demonstrates that you can change the class name and still preserve the mapping to the table.

Instead of pressing Enter, you can click somewhere else in the dialog box, just don't leave the cursor in the field. The Generate Java wizard window should look like this:



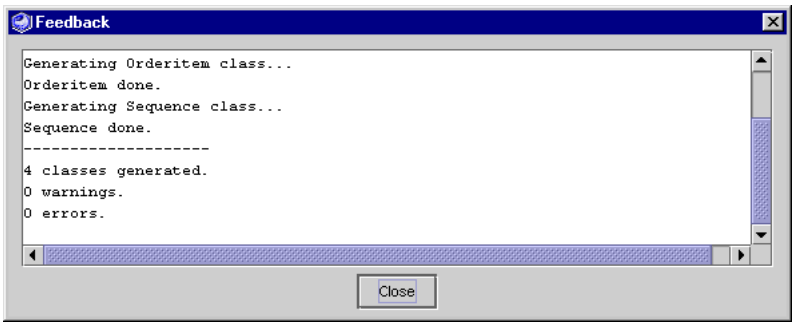

4 Click Next.

The wizard displays a confirmation window that it will generate the classes you want.



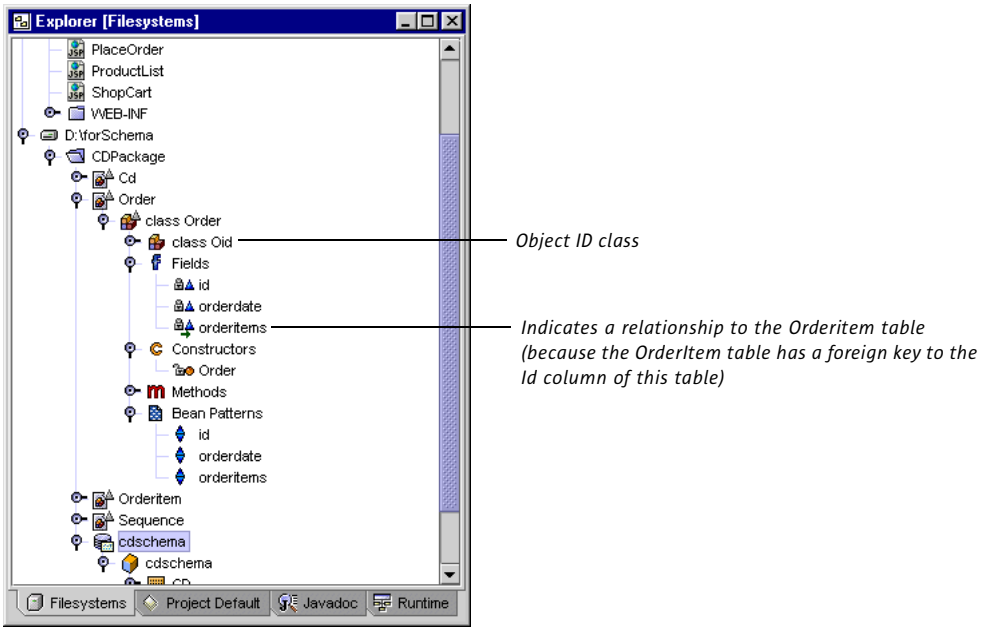
5 Click Finish.

The IDE displays a window that the classes were generated.




6 Click Close to close the message window.

7 In the Explorer, open one of the newly generated classes and observe that the fields were generated from the table's columns and that each has `get` and `set` methods.




Each generated class is associated with an Oid (object ID) class. This is used to identify an instance of a persistence-capable class uniquely. Each instance of the persistence-capable class is associated with an instance of the Oid class that holds its identifier.

8 Compile `CDPackage` by right-clicking it and choosing Compile.

You are ready to enhance the persistence-capable classes.

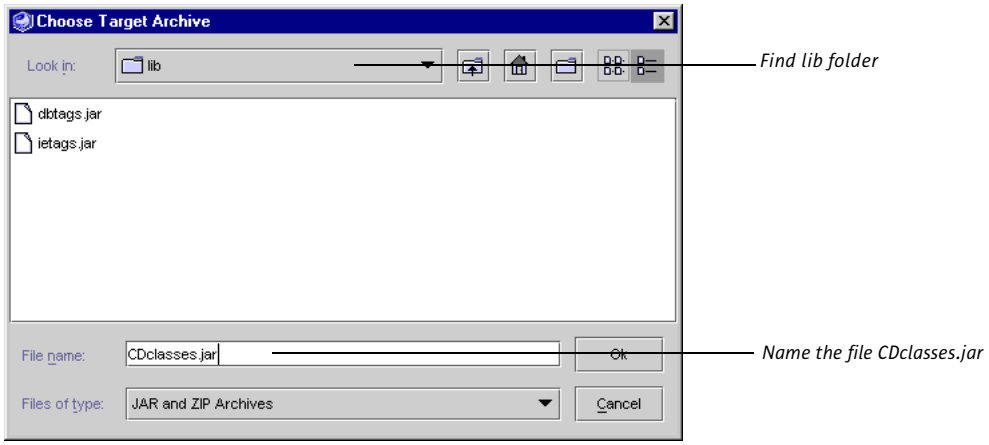
## Enhance the Persistence-Capable Classes

For a class to use the Transparent Persistence capabilities, it must implement the `com.sun.forte4j.persistence.PersistenceCapable` interface. This interface allows persistence-capable classes to interact with the runtime environment and declares a set of methods that allow you to check and reset the status of instances of these classes. Code that implements the PersistenceCapable interface is generated by the process known as enhancing the class.

You can enhance persistence-capable classes either by packaging them in a JAR file or by running them in the Forte for Java IDE. For the CDShopCart application, package the `CDPackage` package into a JAR file in the `CDShopCart` web module.

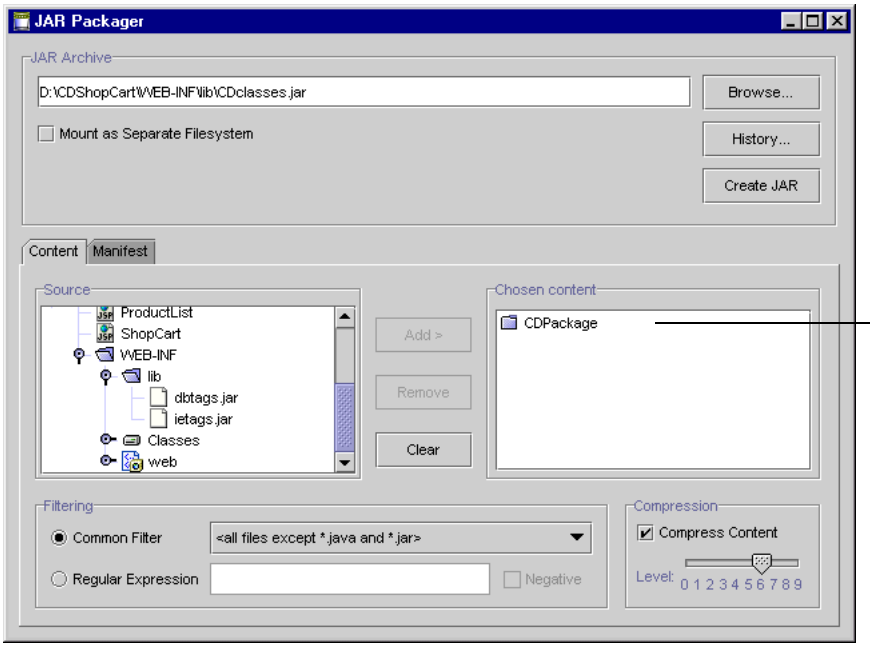
➤ **To enhance the persistence-capable classes:**

1. Right-click on `CDPackage` and choose Tools > Add to JAR to display the JAR Packager wizard.
2. Click the browse button for the field at the top of the wizard window to display the Choose Target Archive dialog box.
3. Choose the `/CDShopCart/WEB-INF/lib` folder and enter `CDclasses.jar` for the class name, as shown:



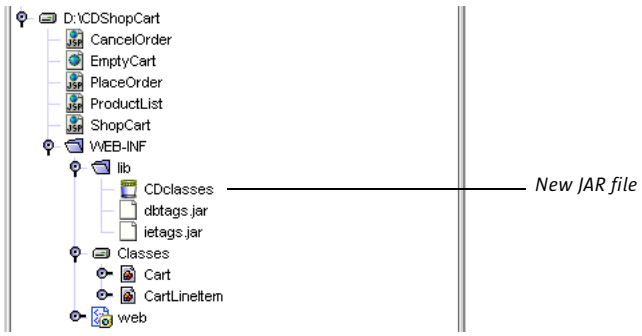

4. Click OK to apply this information.

5 In the JAR Packager wizard, make sure `CDPackage` is listed as the chosen content.




6 Click Create JAR to create the JAR file and close the JAR Packager dialog box.

The `CDPackage` JAR file appears in the Explorer in the `WEB-INF/lib` folder:




You must now remove the `forSchema` Filesystem, so that the `CDPackage` package will not be in the CLASSPATH. When it is removed and you run the application, the IDE executes the enhanced persistence-capable classes, not the unenhanced classes in the `CDPackage` package.

7 To remove the `forSchema` Filesystem, select it and choose File > Unmount Filesystem.

The forSchema Filesystem is removed from the Explorer (and from the CLASSPATH).

You are now ready to create the bean that encapsulates the Transparent Persistence, the `CheckOutBean` bean.

# Creating the Persistence-Aware Bean

In this section, you create the component that encapsulates the Transparent Persistence functionality. This is the `CheckOutBean` bean. It creates a Persistence Manager Factory to specify the properties of the database connection. It then creates a Persistence Manager. You use the Persistence Manager's methods to create `transaction` objects and `query` objects. You use the `query` class's methods to perform the database query functions.

## Create the CheckOutBean

Create the `CheckOutBean` bean with the following general procedures:

1. "Create the Bean and Initialize the Persistence Manager Factory and the Persistence Manager," which follows.
2. "Create a Method to Fetch a CD Based on an ID" on page 83.
3. "Create a Method to Add an Order and Line Items for Each Item in the Cart" on page 85
4. "Add a Method to Get a Sequence Number for the Next Order" on page 87.

## Create the Bean and Initialize the Persistence Manager Factory and the Persistence Manager

In a larger application, you would create a Persistence Manager Factory at the application level, and then create a Persistence Managers for each session. For this small illustration, you create both the Persistence Manager Factory and the Persistence Manager in a session.

➤ **To create the CheckOutBean bean:**

1. Open the `WEB-INF` folder of the `CDShopCart` web module, right-click the `Classes` folder, and choose New > Beans > Bean.
2. In the New From Template New Object Name dialog box that appears, type in **`CheckOutBean`** and click Finish.
3. When a message is displayed prompting you to add the new bean to the project, click **Yes**.

   The new `CheckOutBean` bean is displayed in the Explorer, and its code in the Editor.
4. In the Editor, type the following code to add three new import statements:

```
import com.sun.forte4j.persistence.*;
import java.util.*;
import CDPackage.*;
```

   You need the `com.sun.forte4j.persistence` library to use Transparent Persistence, the `java.util` library because you will use a Collection, and `CDPackage` for the persistence-capable classes you just created.

5 Delete the first three statements of class `CheckOutBean`, all of which begin with the modifier "`private`," and replace them with the following code that declares a Persistence Manager object.

```
private PersistenceManager pm;
```

6 Replace the code from the `CheckOutBean`'s constructor, and add the following code to initialize the Persistence Manager Factory.

CheckOutBean's constructor:
Instantiates a new Persistence Manager Factory:
Specify the JDBC connection:
(Do not break this line!)

```
public CheckOutBean() {
    PersistenceManagerFactory pmf = new
PersistenceManagerFactoryImpl();
    pmf.setConnectionUserName("PUBLIC");
    pmf.setConnectionPassword("PUBLIC");
    pmf.setConnectionDriverName(
              "com.pointbase.jdbc.jdbcUniversalDriver");
    pmf.setConnectionURL("jdbc:pointbase://embedded/cdshopcart");
    pmf.setOptimistic(false);
```

In the above code, use the Persistence Manager Factory's methods (`setConnectionUserName`, and so forth) to specify the database connection to be used by the Persistence Manager.

If you are using a different database, replace the values above for the standard JDBC connect string for your database.

The `setOptimistic` statement specifies concurrency control to be default for all Persistent Manager transactions.

7 Type the following code to the constructor just below the code you just entered to create the Persistence Manager:

```
this.pm = pmf.getPersistenceManager();
```

**For more information** Information on Persistent Manager Factory creation and methods, as well as Persistence Manager creation and methods is provided in the book *Programming Persistence*, in the Javadoc (in the Javadoc pane of the Explorer), and in online help – in the Programming with Persistence-Capable Classes folder under Transparent Persistence.

## Create a Method to Fetch a CD Based on an ID

In this section, create a `getCD` method that instantiates a `Query` object and uses its methods to fetch a CD based on the `Id` passed by the PlaceOrder page.

➤ **To use query methods to fetch a CD:**

1 Delete all of the remaining default code in the `CheckOutBean`.

2 In the Explorer, right-click on `CheckOutBean`'s `Methods` folder and choose New Method.

3 In the New Method dialog box, enter the following values:

Method –
Name: **getCd**
Return type: **Cd**
Access: **public**

Parameter –
Type: **long**
Name: **id**

4 Click OK to create the method.

5 In the Editor, type the following code in the method body:

```
public Cd getCd(long id) {
    Query q = this.pm.newQuery();
    q.setClass(Cd.class);

    q.setCandidates(pm.getExtent(Cd.class, false));

    q.setFilter("id == CDid");

    String param = "Long CDid";

    q.declareParameters(param);

    Collection result = (Collection)q.execute(new Long(id));

    Iterator i = result.iterator();
    Cd theCd = null;
    if (i.hasNext()) {
        theCd = (Cd)i.next();
    }
    return theCd;

}
```

Start here: Create a query.

Bind result class (Cd) to query instance.

Define the input collection for the query.

Bind the query filter to the query instance.

Define a parameter for the placeholder.

Bind the parameter to the query statement.

Execute the query and return the filtered collection.

Iterate through the result to find the specified CD.

Return the specified CD to the caller.

There are three required statements in any query:

- The class of the result (set by the `setClass` method)
- The collection of candidate instances (set by the `setCandidates` method)

  This is either a `java.util.Collection`, or an extent of instances in the data store
- The query filter (set by the `setFilter` method)

  Here `id` is a field in the persistence-capable class `CD`, where `CDid` denotes the query parameter name.

The net effect of these Query methods is approximately the same as the SQL statement `SELECT * FROM CD WHERE ID = ?`.

**For more information on Query methods** See the book *Programming Persistence* for a description of the methods of the Query interface. For brief descriptions, find the Query interface under the Persistence folder in the Javadoc tab of the Explorer.

## Create a Method to Add an Order and Line Items for Each Item in the Cart

In this section, create the `checkout` method to perform the following tasks:

1. Start a transaction.
2. Get an order sequence number and associated date from the CdOrder table.
3. Generate line item and product numbers for each line item and product in the Cart order passed to the method.
4. Commit this data to the database.
5. Return the order sequence number.

➤ **To create the checkout method:**

1. In the Explorer, right-click the `CheckOutBean`'s `Methods` folder and choose New Method.
2. In the New Method dialog box, type the following values:

   Name: **checkout**
   Return type: **int**
   Access: **public**

   This method needs to take the Cart order as its parameter.
3. Click the Add button under the Method Parameters pane and add the following parameter:

   Type: **Cart**
   Name: **myCart**
4. Click OK to create the method.

5 Type the following code into the body of the checkout method to start the transaction:

Use a Persistence Manager method to create the transaction.
Begin the transaction.

```
Transaction tx = pm.currentTransaction();


tx.begin();
```

6 Type the following code to create the order:

Get the next Order sequence number.
Create a new order.
Set the primary key.
Set the current date.
Tell the Persistence Manager to mark for database update.

```
int ordNum = this.getSequenceNumber("CDORDER", 1);

Order ord = new Order();
ord.setId(ordNum);
ord.setOrderdate(new Date());
pm.makePersistent(ord);
```

Next, for each item in the cart, add code that adds a line item to the order.

7 Type the following code next:

Initialize the Line Item number.
Create a new hash list to store all Line Items.
Next item in the cart.
Create a new line item.
Set the primary key.
Add to the collection.
Tell the Persistence Manager to mark for database update.

```
int itemNum = 1;
HashSet itemList = new HashSet();

Iterator i = myCart.lineItems.iterator();
while (i.hasNext()) {
    CartLineItem c = (CartLineItem)i.next();
    Orderitem item = new Orderitem();
    item.setOrderid(ordNum);
    item.setLineitemid(itemNum++);
    item.setCd(getCd(c.getId()));
    itemList.add(item);
    pm.makePersistent(item);

}
```

8 Then, type this code to update the order:

Update the ORDER to contain all the line items.

```
ord.setOrderitems(itemList);
```

9 Finally, type this code to commit this transaction and return the order number:

```
tx.commit();
return ordNum;
```

## Add a Method to Get a Sequence Number for the Next Order

This is the final method to create in the `CheckOutBean`.

➤ **To create the getSequenceNumber method:**

1. In the Explorer, right-click on `CheckOutBean`'s `Methods` folder and choose New Method.
2. In the New Method dialog box, enter the following values:

   Method –
   Name: **getSequenceNumber**
   Return type: **int**
   Access: **public**

   Parameter –
   Type: **java.lang.String**
   Name: **tableName**
3. Click OK to create the parameter, and click the Add button again.
4. Add the following parameter:

   Type: **int**
   Name: **amount**
5. Click OK to create the method.
6. Type the following code in the body of the method to initialize the key:

```
int key = 0;
```

7. Then, type the following code to query the database to retrieve the sequence number for the Cart order stored there.

   The `sequenceQuery` within the `getSequence()` method generates a new primary key for the given table. This provides unique primary keys for the CDShopCart application.

Create a new query.

Bind the result class (Sequence) to the query instance.

Define the input collection.

The filter: `tablename` is the field and `name` is the parameter name.

Define the parameter.

Bind the parameter to the query statement.

```
Query sequenceQuery = pm.newQuery();
sequenceQuery.setClass(Sequence.class);

sequenceQuery.setCandidates(pm.getExtent(Sequence.class, false));
sequenceQuery.setFilter("tablename == name");


String param = "String name";
sequenceQuery.declareParameters(param);
```

Execute the query and return the filtered collection.

```
Collection result = (Collection)sequenceQuery.execute(tableName);

Iterator i = result.iterator();
Sequence s = (Sequence)i.next();
```

Allow the Persistence Manager to update the next key.

```
key = s.getNextpk().intValue();

s.setNextpk(new Integer(key + amount));
```

Return key to caller.

```
return key;
```

Note

If you are using an Oracle database, use a `Long` instead of `Integer` in the next-to-last line.

**8** Right-click on the `CheckOutBean` bean and choose Compile to compile the bean.

# Modifying the PlaceOrder Page to Call CheckOutBean

The only task left is to modify the PlaceOrder JSP page to call the `CheckOutBean` bean and display the order number that it returns. Do this by adding the following elements:

1. `usebean` tags to allow the `Cart` and `CheckOutBean` beans to be used by the page.
2. Declare an order number.
3. Set the order number to the one returned by the `CheckOutBean`.
4. Modify the displayed message to include the order number.

➤ **To modify the PlaceOrder JSP page:**

1. Under the H1 heading tag, add the following usebean tags:

Use the Cart bean.
Use the CheckOutBean bean.

```
<jsp:useBean id="myCart" scope="session" type="Cart" />
<jsp:useBean id="checker" scope="session" class="CheckOutBean" />
```

2. Under these statements, and within the Java scriptlet, add the two noted statements:

Add this declaration:

Set the ordNum to the order number returned by the checkout method.

```
<%! int ordNum; %>

<%
ordNum = checker.checkout(myCart);


session.invalidate();
%>
```

3. And finally, modify the displayed message to use the ordNum, changing the one line into two lines:

```
Your order has been placed. For future reference, your order
number is <%=ordNum%>.

Thank you for shopping.
```

4. Right-click on the PlaceOrder JSP page and choose Compile.

# Test Running the New CDShopCart Application

As with the basic CDShopCart application you tested (see "Test Run the Three Message Pages" on page 68), test run the application by running the ProductList page, adding CD items to the Shopping Cart, and then performing various actions that result in displaying various pages. Here, you choose the specific actions that place an order. This calls the `CheckOutBean` under the covers, but the results—the generated order number in the PlaceOrder page's message—prove that the `CheckOutBean` did its work.

➤ **To test the message pages:**

1. Select the `CDShopCart` web module and choose Build > Build.

   Everything should compile successfully.

2. Right-click the ProductList JSP page and choose Execute (restart server).

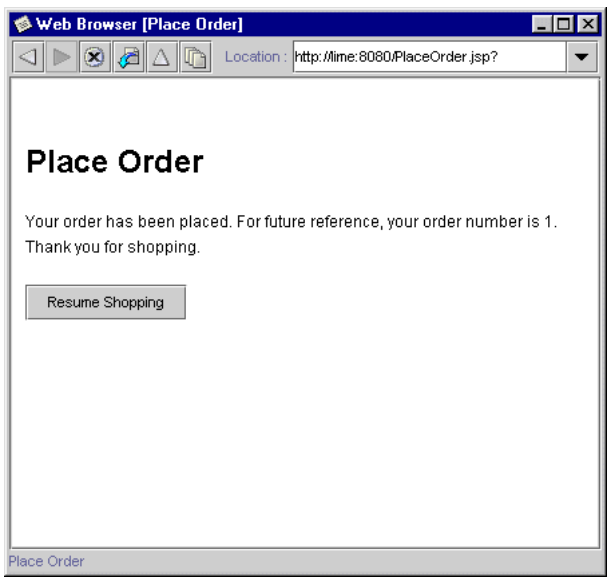
   After a few seconds, the CD Catalog List page is displayed.

3. Click one of the Add buttons to navigate to the Shopping Cart page.

4. Click the Resume Shopping button to return to the CD Catalog List page.

5. Add more CDs to the cart as you wish, and then test the Place Order button.

6. When the Place Order page appears, it should look something like this:




Congratulations! You have completed the CDShopCart tutorial!

# Index

# D



# E



# F



# H



# I

# J



# L



# M



# N



# O



# P

# Q



# R



# S

# T



# W