Forte™ for Java™, Community Edition Tutorial

Forte for Java, Community Edition, 3.0

Sun Microsystems, Inc.
901 San Antonio Road
Palo Alto, CA 94303-4900 U.S.A.
650-960-1300

Part No. 816-1408-10
August 2001, Revision A

Send comments about this document to: `docfeedback@sun.com`

Please
Recycle

Adobe PostScript™

Contents

Figures

Tables

Preface

Welcome to the Forte™ for Java™, Community Edition tutorial. In this tutorial, you will learn how to use the features introduced in the Community 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.

You can create this tutorial on the following platforms and operating systems:

- Solaris™ 8 *SPARC™ Platform Edition*
- Solaris 7 *SPARC Platform Edition*
- Microsoft Windows 2000
- Microsoft Windows NT 4.0, SP6
- Red Hat Linux 6.2

All screen shots in this book are from the Windows NT version of the Forte for Java software. You should have no trouble translating the slight visual differences to other platforms. Although almost all procedures use the Forte for Java user interface, occasionally you might be instructed to enter a command at the command line. In such cases, examples are given with the prompt and syntax for a Microsoft Windows command window. For example:

```
c:\>cd MyWorkDir\MyPackage
```

To translate for UNIX® or Linux environments, simply change the prompt and use forward slashes:

```
% cd MyWorkDir/MyPackage
```

Before You 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, Community 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

This book requires a knowledge of J2EE concepts, as described in the following resources:

- Java 2 Platform, Enterprise Edition Blueprints `www.java.sun.com/j2ee/blueprints`
- *Java 2 Platform, Enterprise Edition Specification*—`www.java.sun.com/products`
- *Java 2 Enterprise Edition Developer's Guide*— `www.java.sun.com/j2ee/j2sdkee/devguide1_2_1.pdf`
- *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`

How This Book Is Organized

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

Chapter 1 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 describes the architecture of the tutorial application.

Chapter 3 provides step-by-step instructions for creating the tutorial application, a simple online shopping cart application for the purchase of music CDs.

Chapter 4 describes how to extend the tutorial application to enable writing customer order data to a database. You use Transparent Persistence technology to accomplish this.

Typographic Conventions

Typeface	Meaning	Examples
`AaBbCc123`	The names of commands, files, and directories; on-screen computer output	Edit your `.login` file. Use `ls -a` to list all files. `% You have mail`.

`AaBbCc123`	What you type, when contrasted with on-screen computer output	`%` **`su`**
		`Password:`

AaBbCc123	Book titles, new words or terms, words to be emphasized	Read Chapter 6 in the *User's Guide*. These are called *class* options. You *must* be superuser to do this.
AaBbCc123	Command-line variable; replace with a real name or value	To delete a file, type `rm` *filename*.

Related Documentation

Forte for Java documentation includes books delivered in Acrobat Reader (PDF) format, online help, Readme files of example applications, and Javadoc™ documentation.

Documentation Available Online

The documents in this section are available from the Forte for Java portal, the `docs.sun.com`^{SM} web site, and from Fatbrain.com, an Internet professional bookstore.

The documentation link of the Forte for Java portal is at `http://www.sun.com/forte/ffj/documentation/index.html.` The `docs.sun.com`^{SM} web site is at `http://docs.sun.com`. Fatbrain.com is at `http://www.fatbrain.com/documentation/sun.`

- Release Notes (PDF format)

  Available for each Forte for Java edition. Describe last-minute release changes and technical notes.

- *Getting Started Guide* (PDF format)

  Available for each Forte for Java edition. Describes how to install the Forte for Java product on each supported platform and includes other pertinent information, such as system requirements, upgrade instructions, web server and application server installation instructions, command-line switches, installed subdirectories, Javadoc setup, databases integration, and information on how to use the Update Center.

- The Forte for Java Programming Series (PDF format)

  This series provides in-depth information on how to use various Forte for Java features to develop well-formed J2EE applications.

  - *Building Web Components* - part no. 816-1410-10

    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* - part no. 816-1411-10

    Describes support for different persistence programming models provided by Forte for Java: JDBC and Transparent Persistence.

  - *Building Enterprise JavaBeans Components* - part no. 816-1401-10

    Describes how to build Enterprise JavaBeans components—session beans and entity beans with container-managed or bean-managed persistence—using the Forte for Java EJB Builder wizards and other graphical user interfaces.

  - *Building Web Services* - part no. 816-1400-10

    Describes how to use the tools provided by the Web Services module to build web services. Web Services are application business services published as Extensible Markup Language (XML) documents delivered over HTTP connections.

  - *Building JSP Pages That Use XML Data Services* - part no. 816-1399-10

    Describes how to use the Forte for Java Enterprise Service Presentation Toolkit to incorporate dynamic XML data in HTML.
  - *Assembling and Executing J2EE Modules and Applications* - part no. 816-1402-10

    Describes how to assemble EJB modules and web modules into a J2EE application, and how to deploy and run a J2EE application.
- Forte for Java tutorials (PDF format)

  You can also find the completed tutorial applications in your user settings directory, under `sampledir/tutorial`.
  - *Forte for Java, Community Edition Tutorial* - part no. 816-1408-10

    Provides step-by-step instructions for building a simple J2EE web application using Forte for Java, Community Edition tools.
  - *Forte for Java, Enterprise Edition Tutorial* - part no. 816-1409-10

    Provides step-by-step instructions for building an application using Enterprise JavaBeans components, Web Services technology, and Forte for Java Enterprise Service Presentation Toolkit technology.

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 Microsoft Windows and Linux), or by choosing Help > Contents. Either action displays a list of help topics and a search facility.

Examples

Several examples, with accompanying `Readme` files, that illustrate a particular Forte for Java feature are available in the `sampledir/examples` subdirectory of your user settings directory. In addition, you can download Enterprise Edition-specific examples from the Forte for Java portal and unzip them into the `examples` directory. Completed tutorial applications—including the applications described in *Forte for Java, Enterprise Edition Tutorial* and this document—are in the `sampledir/tutorial` directory.

Javadoc Documentation

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

Accessing Sun Documentation Online

A broad selection of Sun system documentation is located at:

`http://www.sun.com/products-n-solutions/hardware/docs`

A complete set of Solaris documentation and many other titles are located at:

`http://docs.sun.com`

Ordering Sun Documentation

Fatbrain.com, an Internet professional bookstore, stocks select product documentation from Sun Microsystems, Inc.

For a list of documents and how to order them, visit the Sun Documentation Center on Fatbrain.com at:

`http://www.fatbrain.com/documentation/sun`

Sun Welcomes Your Comments

Sun is interested in improving its documentation and welcomes your comments and suggestions. You can email your comments to Sun at:

`docfeedback@sun.com`

Please include the part number (816-1408-10) of your document in the subject line of your email.

CHAPTER 1

Getting Started

This chapter explains what you need to do before starting the Forte for Java, Community Edition tutorial. For your convenience, it duplicates some installation information from the *Getting Started Guide*. The topics covered in this chapter are:

- "Software Requirements for the Tutorial," which follows
- "Starting the Forte for Java IDE" on page 9
- "Understanding the Forte for Java Directory Structure" on page 12
- "Creating the Tutorial Database Tables" on page 13

Software Requirements for the Tutorial

This section describes how to prepare your system before starting the Forte for Java, Community 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.

You can access general system requirements from the release notes or from the Forte for Java portal's Documentation page `http://www.sun.com/forte/ffj/resources/documentation/index.html`.

What You Need to Run the Forte for Java IDE

The Forte for Java IDE requires the Java Development Kit. When you install the IDE, the installer searches your system for the JDK™ software and will notify you and stop the installation if the correct version is not installed on your system. You can download the correct version of the JDK from the Forte for Java portal.

What You Need to Create and Run the Tutorial

You need the following items to create and run the tutorial. Some of these items are included in the default installation of Forte for Java, Community Edition.

- Database software—any of the following:
  - PointBase Network Server, version 3.5

    You can install PointBase when you install Forte for Java, Community Edition. To see that PointBase was installed, look for a `pointbase` directory under the directory where Forte for Java is installed. If PointBase was not installed, you can run the installer again to install it.
  - Oracle 8.1.7, JDBC Thin Driver 8.1.7
  - Microsoft SQL Server 2000, WebLogic JDBC driver 5.1.0 or JDBC-ODBC bridge (SQL Server 2000 ODBC driver)
  - IBM DB2 7.1, JDBC Thin Driver for DB2 7.1
- A web server—either of the following:
  - Tomcat, version 3.2

    The tutorial is a web application, which requires a web server. This tutorial uses an embedded version of Tomcat within the IDE that provides the functionality of a web server for testing purposes.
  - iPlanet Web Server, version 6.0
- A web browser

  You need a web browser to view the tutorial application pages. This can be either Netscape Communicator™ or Microsoft's Internet Explorer. Netscape Communicator, version 4.7x, is available for download from the Forte for Java portal or the product CD.

Starting the Forte for Java IDE

You start the Forte for Java IDE by running the program executable, as described in the following sections, and more fully in the *Getting Started Guide*.

Starting the IDE on Solaris, UNIX, and Linux Environments

After installation, a `runide.sh` script is in *forte4j-home*`/bin` directory. Launch this script by typing:

```
$ sh runide.sh
```

For ways to customize this script, see "Modifying the Session With Command-Line Switches" on page 10.

Starting the IDE on Microsoft Windows

After installation, there are *three ways* to start the IDE:

- Double-click the Forte for Java CE icon on your desktop.

  This icon runs the `runidew.exe` executable, which launches the IDE without a console window. This icon exists in the *forte4j-home*`\bin` directory, along with an alternative icon—`runide.exe`. The `runide.exe` icon launches the IDE 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 list of running threads or Ctrl-C to immediately terminate the program.
- Alternatively, you can launch the IDE by choosing Start > Programs > Forte for Java CE > Forte for Java CE.
- Or you can run any of the executables from the command line.

  For example:

```
C:\> runide.exe [switch]
```

Modifying the Session With Command-Line Switches

TABLE 1-1 describes the switches that you can use to modify how you launch the IDE. This information is also available from the *Getting Started Guide*, but is provided here for your convenience.

- On Microsoft Windows systems

  You can set options when running the IDE on the command line.

- In Solaris, Linux, and other UNIX environments

  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 1-1 `runide` Command-Line Switches

Switch	Meaning
`-classic`	Uses the classic JVM.
`-cp:p` *addl-classpath*	Adds a class path to the beginning of the Forte for Java class path.
`-cp:a` *addl-classpath*	Adds a class path to the end of the Forte for Java class path
`-fontsize` *size*	Sets the font size used in the GUI to the specified size.
`-J`*jvm-flags*	Passes the specified flag directly to the JVM. (There is no space between `-J` and the argument.)
`-jdkhome` *jdk-home-dir*	Uses the specified Java 2 SDK instead of the default SDK.
`-h` or `-help`	Opens a GUI dialog box that lists the command-line options.
`-hotspot` or `-client` or `-server` or `-classic` or `-native` or `-green`	Uses the specified type of JVM.
`-single`	Runs the IDE in single-user mode. Enables you to launch the IDE from *forte4j-home* instead of from your user settings directory.
`-ui` *UI-class-name*	Runs the IDE with the given class as the IDE's look and feel.
`-userdir` *user-directory*	Uses the specified directory for your user settings for the current session. See the next section for more information.

Specifying Your User Settings Directory

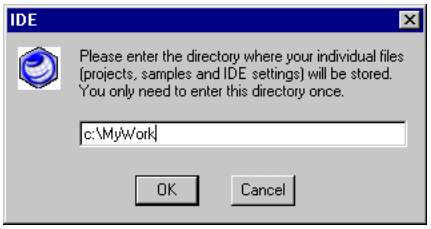
By default, you run the Forte for Java software in multiuser mode, yet store your individual projects, samples, and IDE settings in your own special directory. This enables individual developers to synchronize their development activities, while keeping their own personal work and preferences separate.

- In Solaris, UNIX, or Linux environments

  If you don't explicitly specify a user settings directory with the `-userdir` command-line switch, user settings are located by default in *user-home*`/ffjuser30`.

- On Microsoft Windows systems

  At first launch of the Forte for Java IDE, you are prompted for the specification of this directory. Use a complete specification, for example, `C:\MyWork`.


IDE

Please enter the directory where your individual files (projects, samples and IDE settings) will be stored. You only need to enter this directory once.

c:\MyWork

  You can later specify a different user settings directory by using the `-userdir` command-line switch when launching the IDE.

Understanding the Forte for Java Directory Structure

When you install the Forte for Java software, the subdirectories in TABLE 1-2 are included in your installation directory.

TABLE 1-2 Forte for Java Directory Structure

Directory	Purpose
`beans`	Contains JavaBeans™ components installed in the IDE.
`bin`	Includes Forte for Java launchers (as well as the `ide.cfg` file on Microsoft Windows installations).
`docs`	Contains the Forte for Java help files and other miscellaneous documentation. (Release notes are found under *forte4j-home*.)
`iPlanet`	Contains files used by the iPlanet plug-ins.
`javadoc`	The directory mounted by default in the IDE's Javadoc repository. Both Javadoc provided with the IDE and user-created Javadoc are stored here.
`lib`	Contains JAR files that make up the IDE's core implementation and the open APIs.
`modules`	Contains JAR files of Forte for Java modules.
`pointbase`	Contains the executables, classes, databases, and documentation for the PointBase Network Server database (if installed).
`sources`	Contains sources for libraries that might be redistributed with user applications.
`system`	Includes files and directories used by the IDE for special purposes. Among these are `ide.log`, which provides information useful when seeking technical support.
`teamware`	Contains Forte for Java TeamWare module files (if installed).

When you launch the Forte for Java software in the default (multiuser) mode, the subdirectories in TABLE 1-3 are installed in your user settings directory. Most of them correspond to subdirectories in the Forte for Java home directory, and are used to hold your settings.

TABLE 1-3 Directory Structure for the User Settings Directory

Directory	Purpose
`beans`	Contains user settings for JavaBeans components installed in the IDE.
`javadoc`	Contains user settings for Javadoc files installed in the IDE.
`lib`	Contains user settings for the system `lib` files.
`modules`	Contains modules downloaded from the Update Center.
`sampledir`	The directory mounted by default in the Filesystems pane of the Explorer. Objects you create in the IDE are saved here unless you mount other directories and use them instead.
`sampledir/examples`	Contains several example applications.
`sampledir/tutorials`	Contains several tutorial applications, including the EatersDigest tutorial described in this document and its database scripts.
`system`	Contains user settings for system files and directories.

Creating the Tutorial Database Tables

Before you start the Forte for Java, Community 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.

The provided scripts listed in TABLE 1-4 are located in the `sampledir/tutorial/CDShopCart/SQLscripts` subdirectory of your user settings directory.

TABLE 1-4 SQL Scripts for Creating the Tutorial Tables

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 database schemas shown in TABLE 1-5.

TABLE 1-5 `CDCatalog` Database Tables

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

The CD table contains the records shown in TABLE 1-6.

TABLE 1-6 CD Table Records

ID	CDtitle	Artist	Country	Price
1	Yuan	The Guo Brothers	China	14.95
2	Drums of Passion	Babatunde Olatunji	Nigeria	16.95
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 contains the records shown in TABLE 1-7.

TABLE 1-7 Sequence Table Records

tableName	nextPK
CdOrder	1

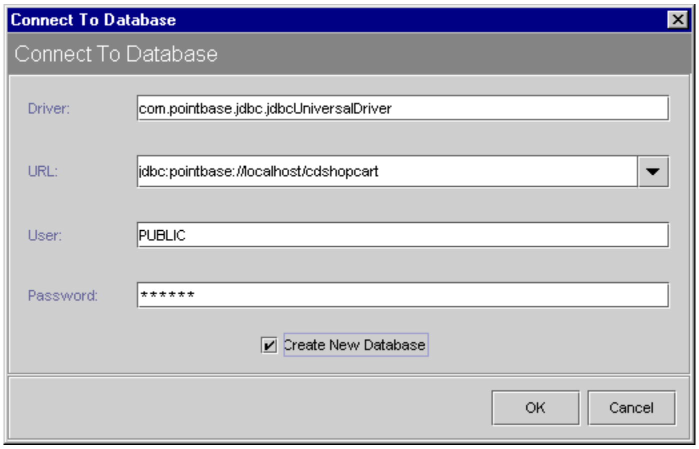
Installing the Tables in PointBase

First create a tutorial database and load the tables into it. (If you prefer, you can install the tables in any PointBase database you choose.)

1. **Start the PointBase Network Server.**
   - In Solaris or Linux environments: Run the `netserver.sh` file in the *forte4j-home*`/pointbase/network` directory.
   - On Microsoft Windows: Choose Start > Forte for Java EE > PointBase > Network Server > Server or double-click the `netserver.bat` file in the *forte4j-home*`/pointbase/network` directory.
2. **Start the PointBase Console.**
   - In Solaris or Linux environments: Run the `console.sh` file in the *forte4j-home*`/pointbase/client` directory.
   - On Microsoft Windows: Choose Start > Forte for Java EE > PointBase > Client Tools > Console or double-click the `console.bat` file in the *forte4j-home*`/pointbase/client` directory.

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

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



Driver: com.pointbase.jdbc.jdbcUniversalDriver
URL: jdbc:pointbase://localhost/cdshopcart
User: PUBLIC
Password: ******
[✓ Create New Database]

4. **Select the Create New Database checkbox and click OK.**

   The PointBase Console is displayed. Wait until the status message ending in "Ready" is displayed before proceeding.

5. **Choose File > Open to display a file browser dialog box.**

6. **Use the file browser to find the** `CDCatalog_pb.sql` **file and click Open.**

   Look in *your-work-dir*`/sampledir/tutorial/CDShopCart/SQLscripts.` The contents of the `CDCatalog_pb.sql` file are copied to the SQL entry window.

7. **Choose SQL > Execute All.**

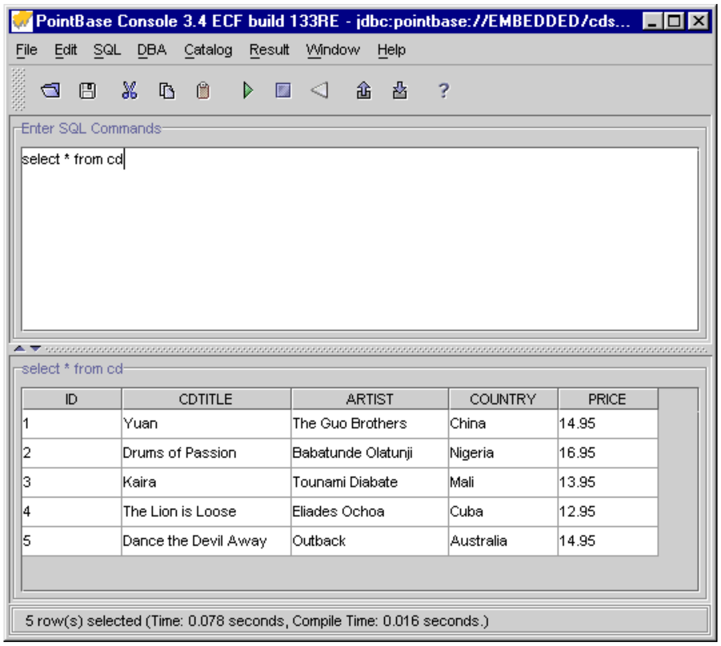
   The message window confirms that the script was executed. (Ignore the initial messages beginning "Cannot find the table…" These appear because there are DROP statements for tables that have not been created yet. These DROP statements will be useful in the future if you want to rerun the script to initialize the tables.)

8. **Test that you have created the table by clearing the SQL entry window (Window > Clear Input) and typing:**

```
SELECT * FROM CD
```

9. **Choose SQL > Execute.**

   Your console should display the CD table.


PointBase Console 3.4 ECF build 133RE - jdbc:pointbase://EMBEDDED/cds...
File Edit SQL DBA Catalog Result Window Help
Enter SQL Commands
select * from cd

```
select * from cd
```

ID	CDTITLE	ARTIST	COUNTRY	PRICE
1	Yuan	The Guo Brothers	China	14.95
2	Drums of Passion	Babatunde Olatunji	Nigeria	16.95
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

```
5 row(s) selected (Time: 0.078 seconds, Compile Time: 0.016 seconds.)
```

10. **Close the PointBase Console window.**

    Proceed to "Starting the Forte for Java IDE" on page 9.

Installing the Tables in Other Databases

If you are not using PointBase for your database, you must first make sure your database is available to the IDE, and then install the tables.

Setting Up a Non-PointBase Database for Forte for Java

To use an alternative database with Forte for Java:

1. **Install the database software on your system.**

   You can use either the server or client version of your database software. If you install the client version, you must also have access to the corresponding database server.

2. **Manually copy the appropriate JDBC-enabled driver into the** *forte4j-home*/`lib/ext` **directory.**

   For example, for Oracle, you'd copy the `classes12.zip` file to this directory.

3. **Specify the correct values for your type of database (and driver) when you create a JDBC connection string in your application code.**

   This is explained in the tutorial, for example in "Using the JDBC connection Tag to Connect to the Database" on page 43.

Installing the Tutorial Tables

To install the tutorial database in an Oracle and Microsoft SQLServer database:

For Oracle:

- **Load the** `CDCatalog_ora.sql` **script from the command line, as shown.**

  In the following example, the tables are loaded into a database on Service Name TUTORIAL with user/password of tutorial/tutorial:

```
c:\>cd your_user_dir\sampledir\tutorial\CDShopCart\SQLscripts
c:\>sqlplus tutorial\tutorial@TUTORIAL @CDCatalog_ora.sql
```

For Microsoft SQLServer:

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

2. **Modify the first line of the** `CDCatalog_ms.sql` **file.**

```
use CDShopCart
```

   Replace `CDShopCart` with the name of your database.

3. **Load the** `CDCatalog_ms.sql` **script.**

   In this example, the tables are loaded into a database on server `MY_MSSQL` with user/password of tutorial/tutorial:

```
c:\>cd your_user_dir\sampledir\tutorial\CDShopCart\SQLscripts
c:\>isql -SMY_MSSQL -Ututorial -Ptutorial < cdcatalog_ms.sql
```

CHAPTER 2

Introduction to the Tutorial

In the process of creating the tutorial example application, you learn how to build components of a web application with Forte for Java, Community Edition features.

This chapter describes the application you will build, first laying out its requirements and then describing an architecture that will fulfill those requirements. The final section describes how you use Forte for Java, Community 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, as follows:

1. The customer selects items from a catalog page to add to a shopping cart
2. The customer can later add more items from the catalog page or remove existing items from the shopping cart.
3. When the customer is ready to make a purchase, the application writes the order to the database, displays a Thank You message and an order number, and ends the session.
4. The customer can then exit the application or return to 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 catalog page and ends when the customer either completes an order or quits the site. The following scenarios 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 a 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.

   The customer clicks the Resume Shopping button on the Shopping Cart page, which causes the application to redisplay the CD catalog page for further selections. 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—there is no Amount column for each CD).

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. (The Cancel Order page is discussed in Scenario 7.)

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 and an order number, 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 at any time 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

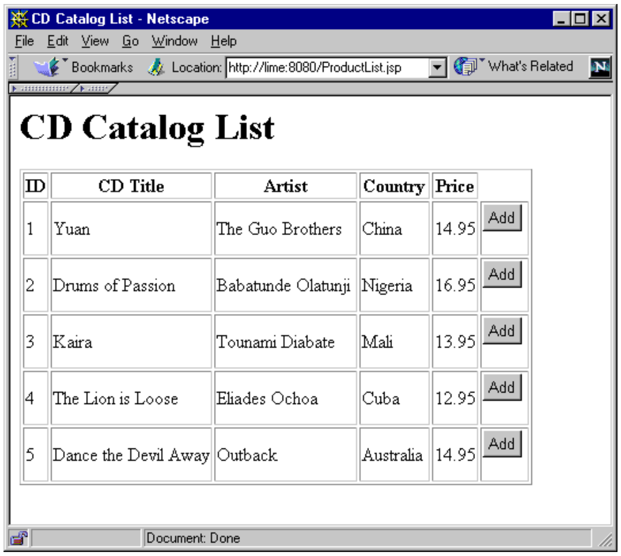
User's View of the Tutorial Application

The user's view of the application illustrates how the scenarios and the functional specification, described in "Functionality of the Tutorial Application" on page 19 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.

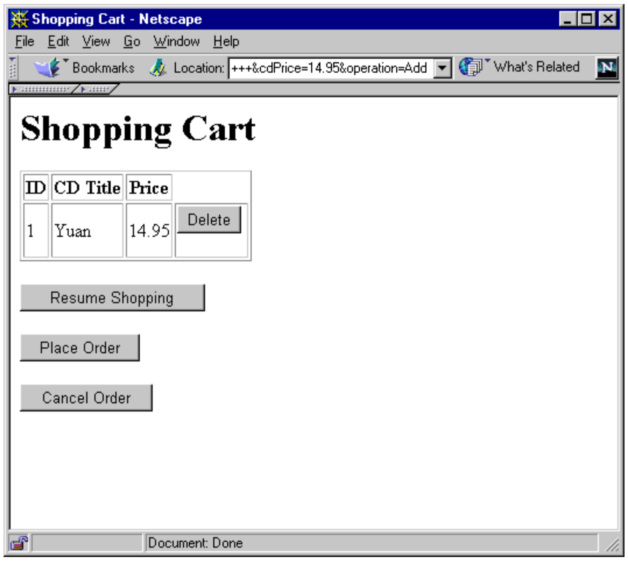

CD Catalog List - Netscape
File Edit View Go Window Help
Bookmarks Location: http://lime:8080/ProductList.jsp What's Related
Document: Done

CD Catalog List

ID	CD Title	Artist	Country	Price	
1	Yuan	The Guo Brothers	China	14.95	Add
2	Drums of Passion	Babatunde Olatunji	Nigeria	16.95	Add
3	Kaira	Tounami Diabate	Mali	13.95	Add
4	The Lion is Loose	Eliades Ochoa	Cuba	12.95	Add
5	Dance the Devil Away	Outback	Australia	14.95	Add

2. **To add a CD to your shopping cart, 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.


Shopping Cart - Netscape
File Edit View Go Window Help
Bookmarks Location: +++&cdPrice=14.95&operation=Add What's Related
Resume Shopping
Place Order
Cancel Order
Document: Done

Shopping Cart

ID	CD Title	Price	
1	Yuan	14.95	Delete

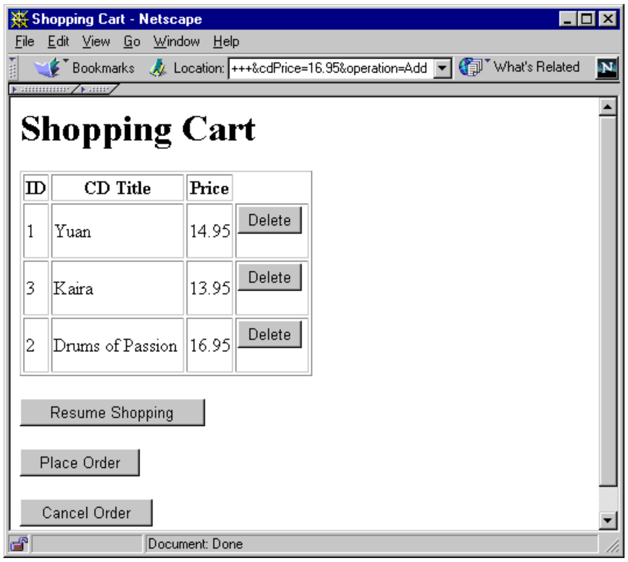
3. **To add another CD, click Resume Shopping, which takes you back to the CD Catalog List page.**

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.


Shopping Cart - Netscape
File Edit View Go Window Help
Bookmarks
Location: +++&cdPrice=16.95&operation=Add
What's Related
Document: Done

Shopping Cart

ID	CD Title	Price	
1	Yuan	14.95	Delete
3	Kaira	13.95	Delete
2	Drums of Passion	16.95	Delete

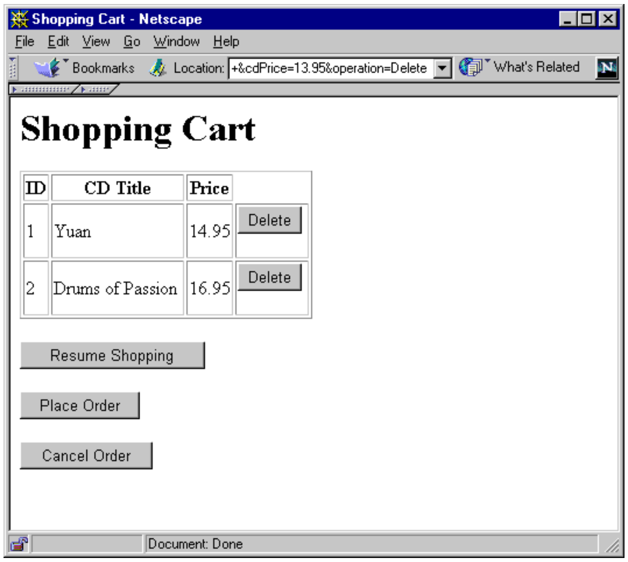
Resume Shopping

Place Order

Cancel Order

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

   The table is redisplayed minus the removed item.


Shopping Cart - Netscape
File Edit View Go Window Help
Bookmarks
Location: +&cdPrice=13.95&operation=Delete
What's Related
Document: Done

Shopping Cart

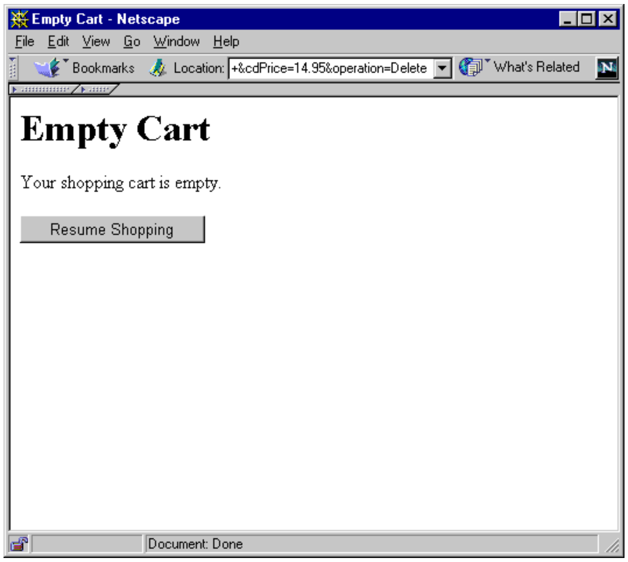
ID	CD Title	Price	
1	Yuan	14.95	Delete
2	Drums of Passion	16.95	Delete

Resume Shopping

Place Order

Cancel Order

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

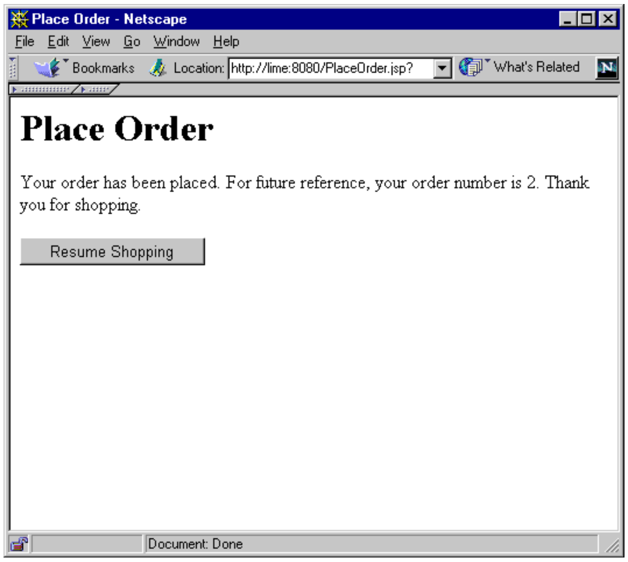

Empty Cart - Netscape
File Edit View Go Window Help
Bookmarks
Location: +&cdPrice=14.95&operation=Delete
What's Related

Empty Cart

Your shopping cart is empty.

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.


Place Order - Netscape
File Edit View Go Window Help
Bookmarks
Location: http://lime:8080/PlaceOrder.jsp?
What's Related

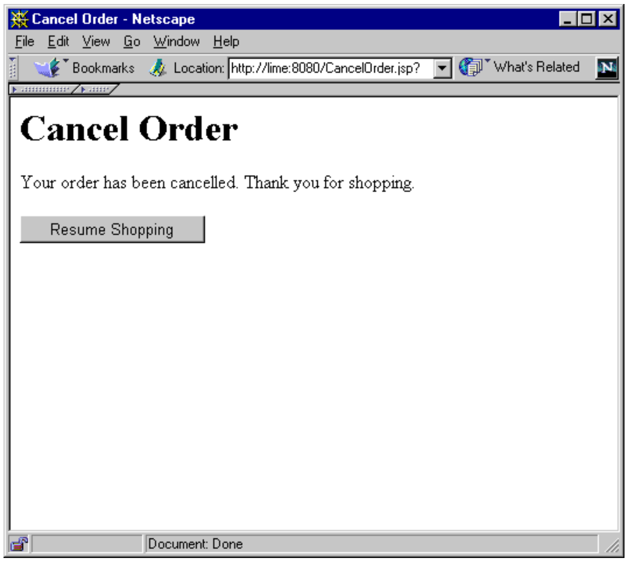
Place Order

Your order has been placed. For future reference, your order number is 2. Thank you for shopping.

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.


Cancel Order - Netscape
File Edit View Go Window Help
Bookmarks Location: http://lime:8080/CancelOrder.jsp?
What's Related

Cancel Order

Your order has been cancelled. Thank you for shopping.

Resume Shopping
Document: Done

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

Architecture of the Tutorial Application

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 a single 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, Community 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 2-1 shows the CDShopCart application elements and their relation to one another.

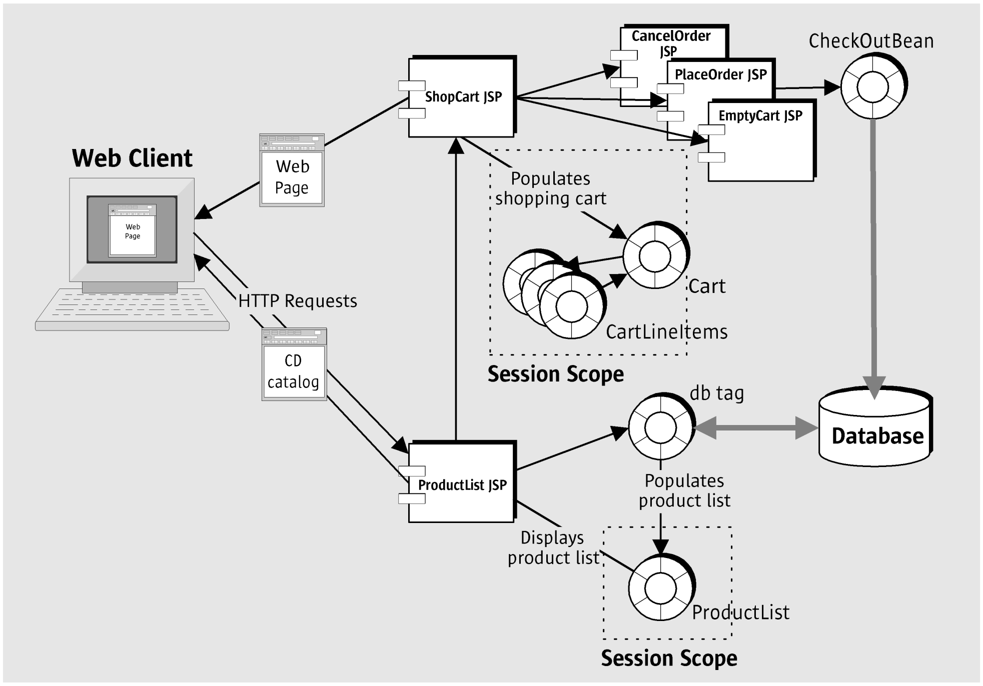


FIGURE 2-1 Architecture of the CDShopCart Application

Application Elements

Briefly, the elements shown in FIGURE 2-1 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 JSP page that displays a message when the customer deletes the last item in the shopping cart. Includes a Resume Shopping button.

- 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 and generate an order number. PlaceOrder displays a message that the order has been placed, provides the order number, 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 and generate a unique order number.

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. 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. `ProductList` uses tags from the Forte for Java database tag library to access the list of CDs from the database, and tags from the Forte for Java presentation library to display them in a table. `ProductList` 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, then uses tags from the Forte for Java presentation tag library to display them in a table. `ShopCart` provides a Delete button on each cart item. When this button is clicked, 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, `ShopCart` forwards to the `EmptyCart` page. `ShopCart` 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 page

  This JSP page is called when the user clicks the Cancel Order button on the `ShopCart` page. This page invalidates the session, displays a message that the order is cancelled, and provides a Resume Shopping button to return to the `ProductList` page.

- `EmptyCart` JSP page

  This JSP page is called when the user removes the last item from the `ShopCart` page. `EmptyCart` displays a message that the cart is empty and includes a Resume Shopping button.

- `PlaceOrder` JSP page

  This JSP page is called when the user clicks the Place Order button on the `ShopCart` page when there are items in the cart. `PlaceOrder` uses the `CheckOutBean` to save the order to the database, then invalidates the session, and displays a Thank you 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 the Forte for Java software installed and set up to run, and the tutorial database tables installed, as described in Chapter 1.

Creating the Basic Application

In Chapter 3 you learn how to use the following Forte for Java, Community Edition 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

The Forte for Java IDE provides tools for automatically creating the hierarchical directory structure of a web application. This structure 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 33 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.

The section "Using Forte for Java Custom Tags" on page 37 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.

The ShopCart JSP page displays CD data passed to it by the ProductList JSP page. The section "Adding 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) and three JSP pages (CancelOrder, PlaceOrder, and EmptyCart).

The section "Creating the CartLineItem JavaBeans Component" on page 50 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 "Creating the Cart JavaBeans Component" on page 55, you learn how to create a bean whose object holds the accumulated line items that have been selected. The Cart bean has methods for adding and removing line items from the cart.

In "Creating the Empty Cart Page" on page 64 section, you create a JSP page that displays a message that the cart is empty. This is to avoid displaying an empty form on the Shopping Cart page.

You create two more JSP pages that hold minor logic compared with ProductList and ShopCart. In "Creating the Place Order Page" on page 66, you create a JSP page that thanks the user 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 "Creating the Cancel Order Page" on page 68, you create a similar page that announces that an order is canceled and invalidates the session.

Test Running the Application

Throughout this chapter, you test run each element just after you create it. The IDE 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 you learn to use Transparent Persistence to write the order to the CdOrder table you created in Chapter 1.

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 75.

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 89. 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 the CheckOutBean" on page 97, you modify the PlaceOrder JSP page to use the results from `CheckOutBean`'s 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 *your_user_directory*`/sampledir/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 the Forte for Java software installed and set up to run, and the tutorial database tables installed, as described in Chapter 1.

This chapter is organized under the following topics:

- "Creating a Web Module," which follows
- "Using Forte for Java Custom Tags" on page 37
- "Creating the Shopping Cart Page and Supporting Elements" on page 49
- "Creating the Three Message Pages" on page 64

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, except that Step 9 will not display the order number. That part is covered in Chapter 4.

Creating a Web Module

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

This section shows you how to implement the shopping cart functionality within a web module by using the Forte for Java, Community Edition.

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 versions of some of the objects.

Note – 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.

Creating the `CDShopCart` Web Module

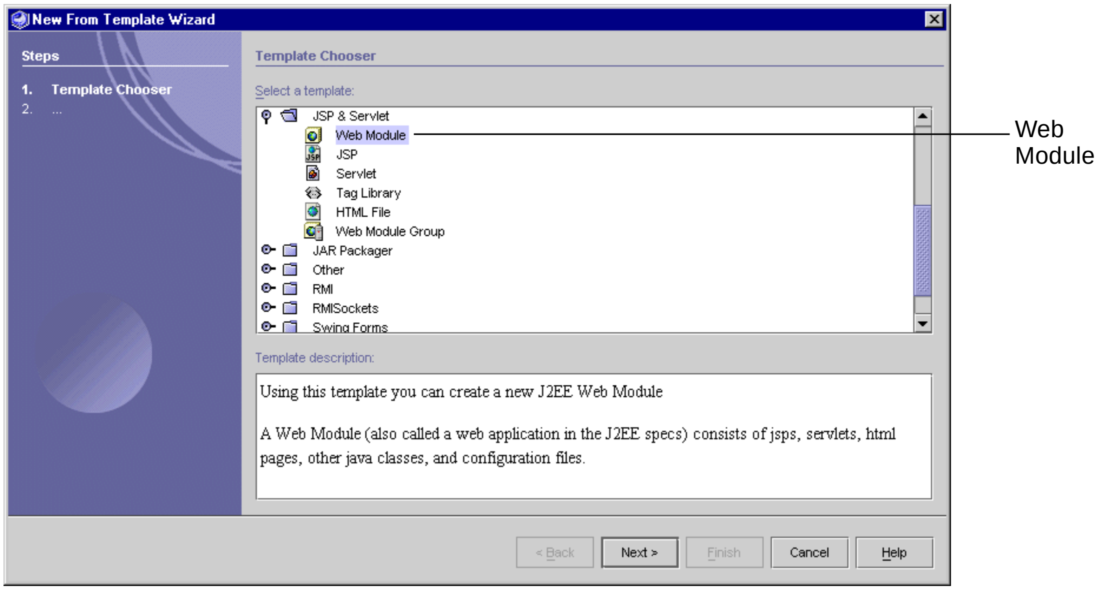
In this section, you create the web module for the CDShopCart application. A web module is a directory. You can use the Forte for Java web module feature to convert an existing directory into a web module, but such a directory doesn't yet exist. You will create one from scratch.

Tip – Before proceeding, you might want to unmount the Filesystem that contains the `sampledir` directory that is provided by default. The `CDShopCart` web module supplied with the Forte for Java software exists in the `sampledir/tutorial` directory. Having two `CDShopCart` web modules can lead to confusion.

To create the CDShopCart web module:

1. **In the Filesystems pane of the Explorer window choose File > New to display the New Template Chooser wizard.**

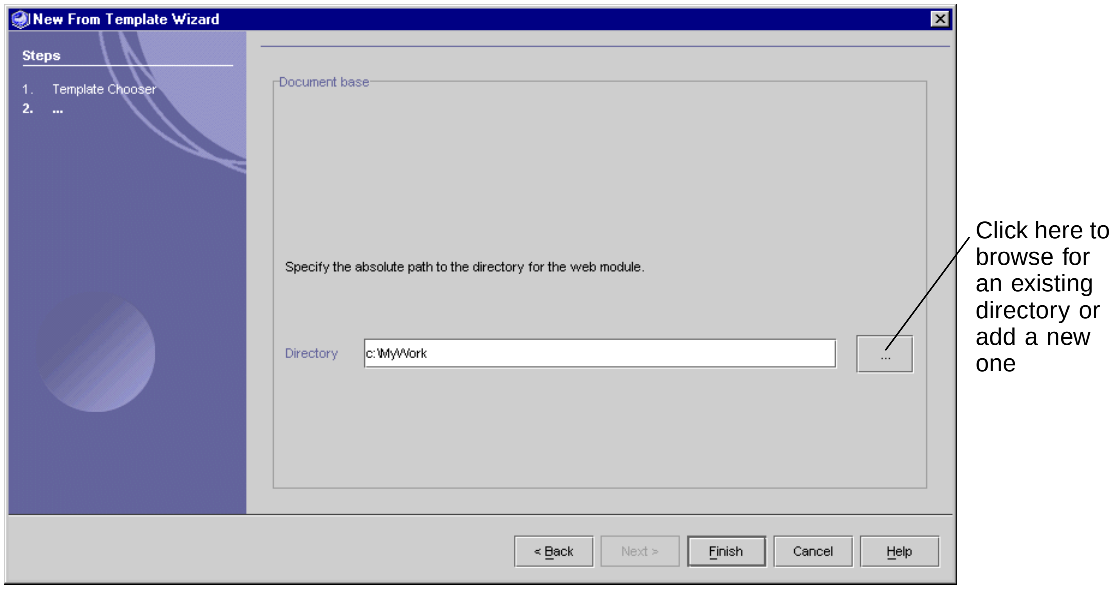
2. **Open the `JSP & Servlet` node and select `Web Module`.**



Using this template you can create a new J2EE Web Module

A Web Module (also called a web application in the J2EE specs) consists of jsps, servlets, html pages, other java classes, and configuration files.

3. **Click Next to display a panel for specifying the path to the directory for the web module.**

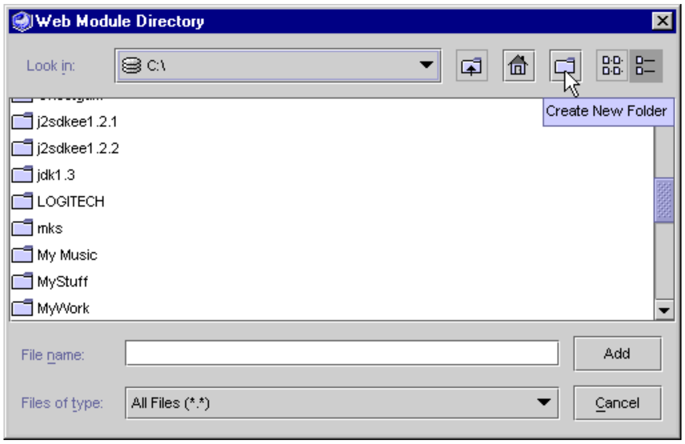


Specify the absolute path to the directory for the web module.

Directory: c:\MyWork

4. **Click the ellipsis 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.**



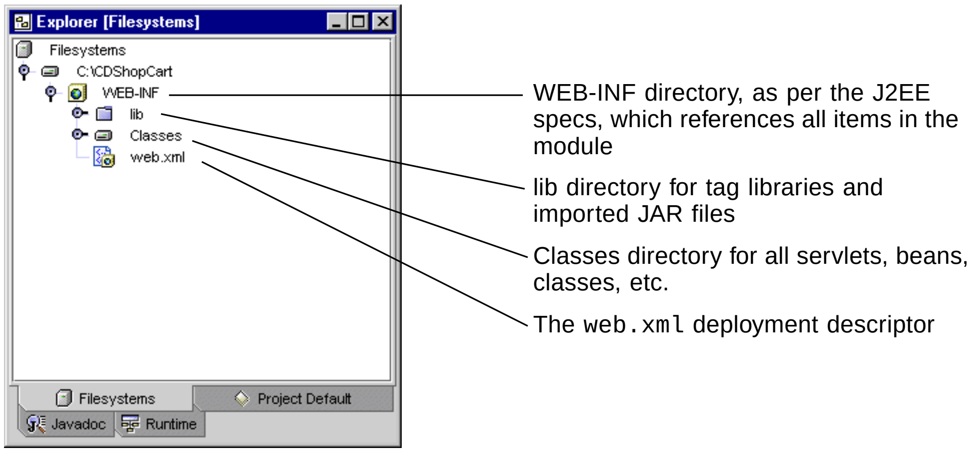
6. **Scroll down to find the folder named New Folder.**
7. **Select the New Folder folder, and click it again to make the name editable.**
8. **Replace the old name with `CDShopCart` and press Enter.**
9. **In the list of folders, find the `CDShopCart` folder, make sure it is selected, and then click 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. **Open the nodes in the web module to reveal what has been created automatically.**



WEB-INF directory, as per the J2EE specs, which references all items in the module

lib directory for tag libraries and imported JAR files

Classes directory for all servlets, beans, classes, etc.

The `web.xml` deployment descriptor

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 Forte for Java 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 summary 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.
  - Tags

    Action elements use XML-style tags as a means of working 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. Java classes associated with each tag implement the tag's functionality.

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

The IDE 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, EJB components, Transparent Persistent classes (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. These source files are for the classes that implement the tag functionality and the XML definitions that describe the tag syntax. They are found in the `jsptaglibs_src.jar` file in the *forte4j-home*`/sources` directory.

Where to Get More Information

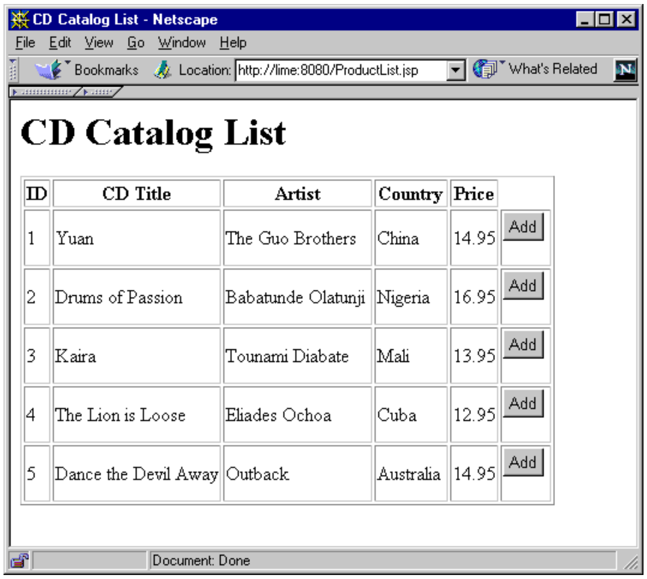
The Forte for Java tag libraries are described in online help, in the Forte for Java Tag Libraries folder under the JavaServer Pages and Servlets main folder. You can find full descriptions of syntax, as well as short examples. A larger demo example is provided in *your_user_dir*`/sampledir/examples/TagLibDemo` 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.

Creating the CD Catalog List Page

This section describes how to create the mechanism for retrieving data from the database you installed in Chapter 1 and displaying it in a table for the user. The page you create looks like FIGURE 3-1.


CD Catalog List - Netscape
File Edit View Go Window Help
Bookmarks
Location: http://lime:8080/ProductList.jsp
What's Related
Document: Done

CD Catalog List

ID	CD Title	Artist	Country	Price	
1	Yuan	The Guo Brothers	China	14.95	Add
2	Drums of Passion	Babatunde Olatunji	Nigeria	16.95	Add
3	Kaira	Tounami Diabate	Mali	13.95	Add
4	The Lion is Loose	Eliades Ochoa	Cuba	12.95	Add
5	Dance the Devil Away	Outback	Australia	14.95	Add

FIGURE 3-1 CD Catalog List Page

Creating this page includes the following tasks:

1. "Adding Forte for Java Tag Libraries to the Web Module" on page 40
2. "Creating the `ProductList` JSP Page" on page 41

3. "Declaring the Tag Libraries" on page 42
4. "Using the JDBC `connection` Tag to Connect to the Database" on page 43
5. "Using the JDBC `query` Tag to Fetch the CD Data" on page 44
6. "Iterating Through the Data With Presentation Tags" on page 45
7. "Creating the Add Button for Each CD Row" on page 46
8. "Cleaning Up With the JDBC `cleanup` Tag" on page 47

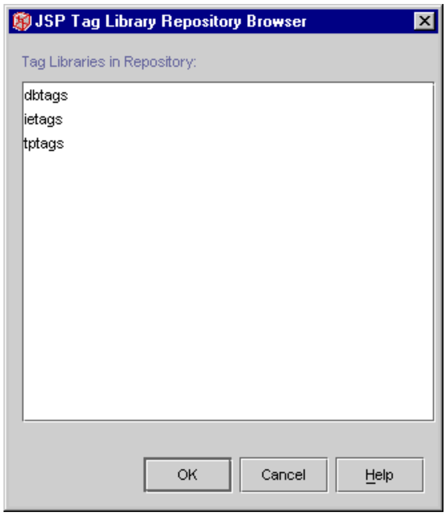
Adding Forte for Java Tag Libraries to the Web Module

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.

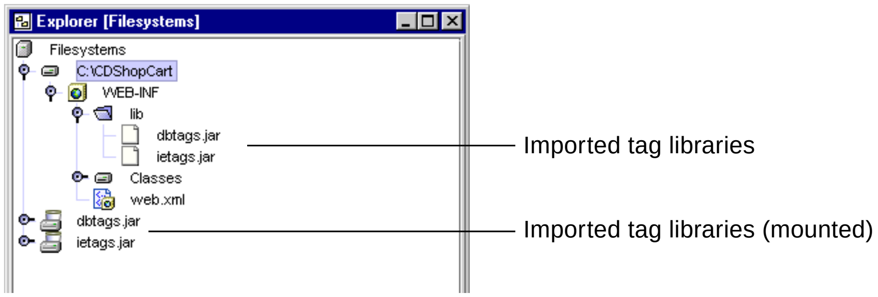

JSP Tag Library Repository Browser
Tag Libraries in Repository:
dbtags
ietags
tptags
OK
Cancel
Help

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, expand the `lib` node under the `WEB-INF` node.

3. **Check that both tag libraries are there.**

   The tag libraries are also separately mounted. The Explorer should look like this:



Tip – Right-click the top level `Filesystems` node and choose Customize. The window that opens displays all the files that are now mounted in your Forte for Java CLASSPATH. You should be able to see the tag libraries there.

Creating the `ProductList` JSP Page

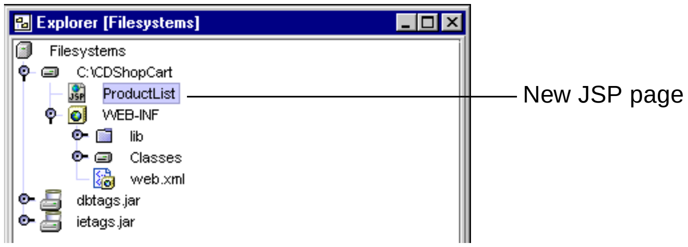
Now, create the page that uses the tags to retrieve the CD data from the database and display it in a table. The title of this page is the "CD Catalog List," and the mechanism that produces it is the `ProductList` JSP page.

1. **In the Explorer, right-click the `CDShopCart` web module and choose New > JSP & Servlet > JSP from the contextual menu.**

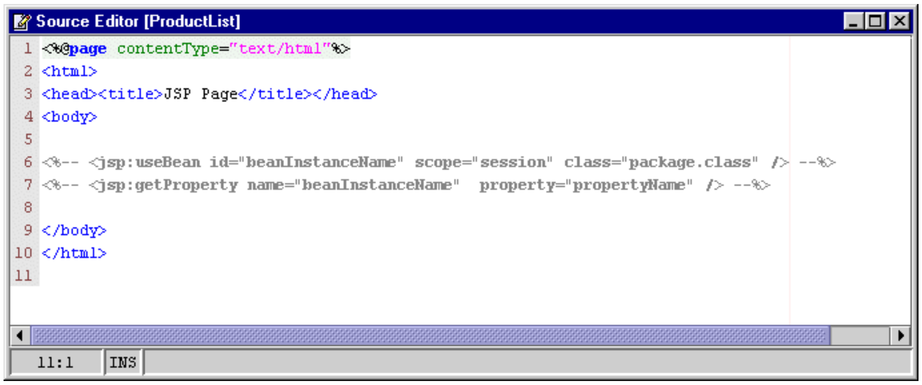
   The New From Template wizard appears displaying the New Object Name panel.

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 Source Editor.



```
<%@page contentType="text/html"%>
<html>
<head><title>JSP Page</title></head>
<body>

<%-- <jsp:useBean id="beanInstanceName" scope="session" class="package.class" /> --%>
<%-- <jsp:getProperty name="beanInstanceName"  property="propertyName" /> --%>

</body>
</html>
```

Note how the Source Editor features color coding and code completion.

Declaring the Tag Libraries

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

The `taglib` directive 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 used 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.

1. **In the body of the** `ProductList` **page, change the document title to** `CD Catalog List`**.**

```
<head><title>CD Catalog List</title></head>
```

2. **Add directives to import the** `dbtags.jar` **and** `ietags.jar` **packages.**

```
//Add a database tag lib directive:
<%@taglib uri="/WEB-INF/lib/dbtags.jar" prefix="jdbc" %>
//Add a presentation tag lib directive
<%@taglib uri="/WEB-INF/lib/ietags.jar" prefix="pr" %>
```

Using the JDBC `connection` Tag to Connect to the Database

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'll use is the JDBC `connection` tag, which is the empty tag type (with lots of attributes). 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, but you will use only the `id`, `driver`, `url`, and `user` attributes, illustrated in the following code:

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

For more information, see the online help on JDBC Data Access Tags, which is found under JavaServer Pages and Servlets, Forte for Java Custom Tag Libraries.

To use a JDBC `connection` tag to connect to the database, first add a page heading (with an H1 tag), then add the `connection` tag below the body HTML tag.

1. **Below the BODY tag in the ProductList page, create a page title.**

```
<body>
<h1> CD Catalog List </h1>
```

2. **Below the header, create the JDBC connection.**

- The following is for a PointBase driver.

```
<jdbc:connection id="jdbcConn"
   driver="com.pointbase.jdbc.jdbcUniversalDriver"
   url="jdbc:pointbase://localhost/cdshopcart"
   user="PUBLIC" password="PUBLIC" />
```

■ If you're using an Oracle database (using the thin driver), use this:

```
<jdbc:connection id="jdbcConn"
   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're using a Microsoft SQLServer database with a Weblogic driver, use this:

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

The default port number for SQLServer is 1433.

Using the JDBC `query` Tag to Fetch the CD Data

Now 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 result set 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, bounded by a start tag and end tag, with attribute/value pairs and an optional body, as shown:

```
<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:

- **In the ProductList page, just following the connection tag, create a query to select all the CD data from the database.**

```
//Use the “jdbcConn connection created above, and put it
//on the session:
<jdbc:query id="productQuery" connection="jdbcConn"
       resultsId="productDS" resultsScope="session" >
    SELECT * FROM CD
</jdbc:query>
```

Iterating Through the Data With Presentation Tags

At this point, create a table, and then fill the table cells with the data. 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 data source 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’s 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"/>
```

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.

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>
```

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

```
//In the following table, retrieve the value from
//each field in the current row of the results.
<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>
//You'll add more here in the next section.
</pr:iterator>
```

Creating 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'll create an HTML form to define the area for user input (clicking the button), and within this area, you'll embed information that will be passed to the Shopping Cart JSP page. 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">
```

2. **Specify the embedded information for product ID, title, and price.**

```
<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"/>">
```

3. **Directly underneath the above code, specify the Add button.**

```
<input type=submit name=operation value=Add>
```

4. **End the form, the cell, and the row.**

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

5. **Below the iterator end tag, end the table.**

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

Cleaning Up With the JDBC `cleanup` Tag

Now add a `cleanup` tag and then end the body of the JSP page. 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.

To clean up resources and end the page:

1. **Type the following code:**

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

The scope for the cleanup tag relates to the scope set for other tags that use the cleanup mechanism. In this case, the `query` tag uses the cleanup mechanism to close the result set that it gets from the query. Because the scope of the result set was to session for the `query` tag (see "Using the JDBC `query` Tag to Fetch the CD Data" on page 44), the `cleanup` tag's scope is also set to session.

2. **Choose File > Save to save your work.**

Test Executing the `ProductList` JSP Page

Now validate your work. Compile the `ProductList` page, then use the integrated Forte for Java runtime system and your web browser to execute the page.

To test execute the `ProductList` 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.

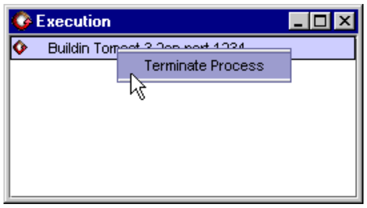
2. **Execute the ProductList JSP by selecting it and clicking the Execute button in the Toolbar (▷).**

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

   The IDE starts the built-in Tomcat server, then switches to the Running 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 3-1.

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

   The process you are terminating is the Tomcat Web Server process.



Note – When you execute the program after the first time in a session, you must choose Execute (Force Reload). This is necessary if you've changed a Java class or Servlet. This isn't necessary if you've only changed a JSP page.

---

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

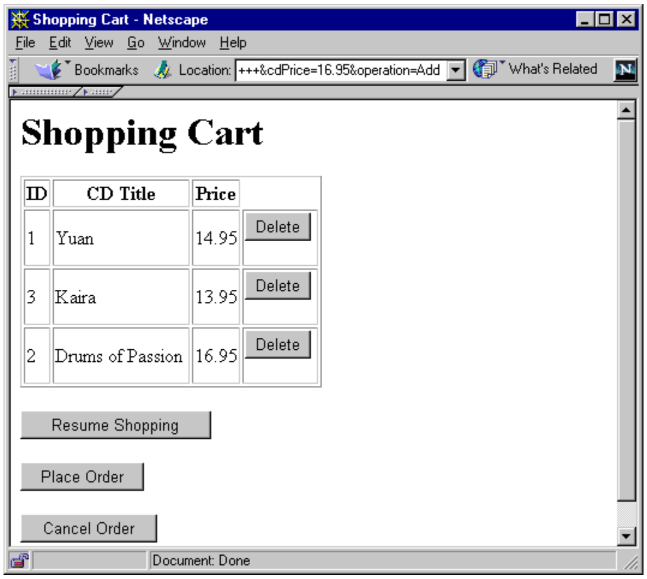
   Congratulations! You've successfully created a JSP page with Forte for Java custom tags to open a connection to a database and retrieve and display data from it. Now create the Shopping Cart page.

Creating the Shopping Cart Page and Supporting Elements

Now create the mechanism for displaying items selected for purchase from the CD Catalog List page. This mechanism includes:

- A bean (`CartLineItem`) to hold the attributes of the selected CD row passed as parameters from the ProductList page
- Another bean (`Cart`) to hold the `CartLineItem` objects
- The `ShopCart` JSP page to receive the `Cart` objects and display them as a row in a table
- A Delete button for each item displayed on the `ShopCart` page
- Cancel Order, Resume Shopping, and Place Order buttons with their implementations

The page you create looks like FIGURE 3-2, when a few items have been selected.


Shopping Cart - Netscape

File Edit View Go Window Help

Bookmarks Location: +++&cdPrice=16.95&operation=Add What's Related

Shopping Cart

ID	CD Title	Price	
1	Yuan	14.95	Delete
3	Kaira	13.95	Delete
2	Drums of Passion	16.95	Delete

Resume Shopping

Place Order

Cancel Order

Document: Done

FIGURE 3-2 Shopping Cart Page

Use the following procedures to create the `ShopCart` page and its beans:

1. "Creating the `CartLineItem` JavaBeans Component" on page 50
2. "Converting the `id` and `price` Properties to Strings" on page 53
3. "Creating the `Cart` JavaBeans Component" on page 55
4. "Creating the Shopping Cart Page" on page 58
5. "Test Executing the Shopping Cart Page" on page 63

Creating the `CartLineItem` JavaBeans Component

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, create three properties on the bean with their accessor methods.

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

   The New From Template wizard appears displaying the New Object Name panel.

2. **In the Name field, type `CartLineItem` and click Finish.**

   The new `CartLineItem` bean is displayed in the Explorer, and its code in the Source Editor. The red badge near the bean ( ) is a "need to compile" indicator. Don't worry about this; you'll compile later.

3. **Open the bean and its class to reveal its contents.**

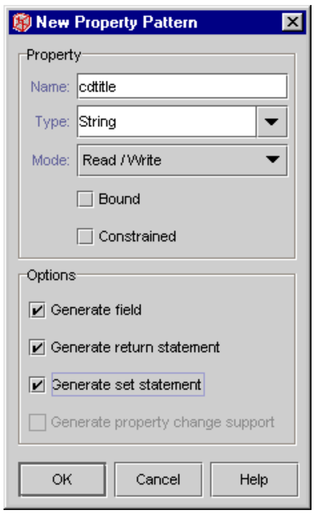
   Now create the `cdtitle` property.

4. **Right-click the `Bean Patterns` node and choose New > Property.**

5. **In the New Property Pattern dialog box, enter the following information:**

   Name: **`cdtitle`**
   Type: **`String`**
   Generate field: **`checked`**
   Generate return statement: **`checked`**
   Generate set statement: **`checked`**

   The dialog box should look like this:



New Property Pattern

Property

Name: cdtitle

Type: String

Mode: Read / Write

☐ Bound

☐ Constrained

Options

☑ Generate field

☑ Generate return statement

☑ Generate set statement

☐ Generate property change support

6. **Click OK to accept the information and close the dialog box.**

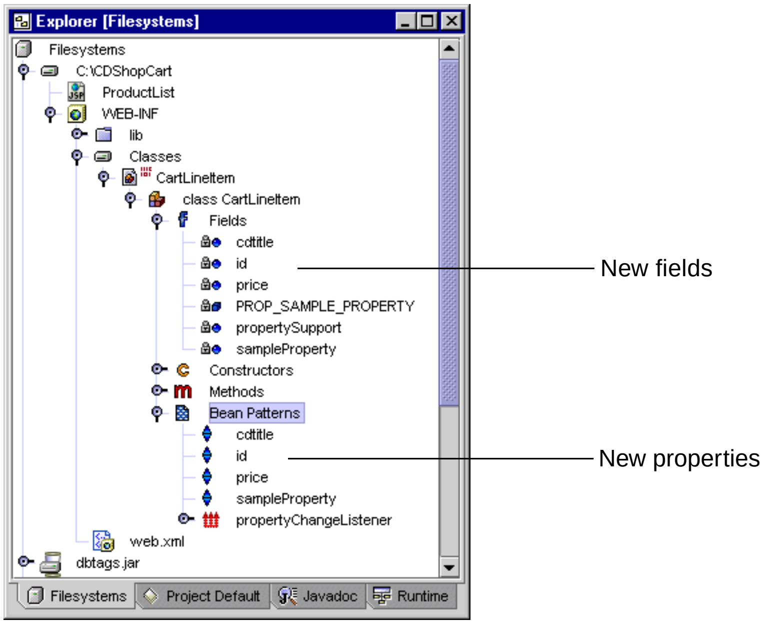
7. **Similarly, create the `id` property with these values:**

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

8. **And finally, create the `price` property with the following values:**

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

9. **Open the `Fields` node (of the `CartLineItem` bean class) to see the new fields you created.**



```
Explorer [Filesystems]
Filesystems
  C:\CDShopCart
    ProductList
    WEB-INF
      lib
      Classes
        CartLineItem
          class CartLineItem
            Fields
              cdtitle
              id
              price
              PROP_SAMPLE_PROPERTY
              propertySupport
              sampleProperty
            Constructors
            Methods
            Bean Patterns
              cdtitle
              id
              price
              sampleProperty
              propertyChangeListener
      web.xml
  dbtags.jar
```

10. **Open the `Methods` node to see the new `get` and `set` methods for each field.**

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

Converting the `id` and `price` Properties to Strings

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

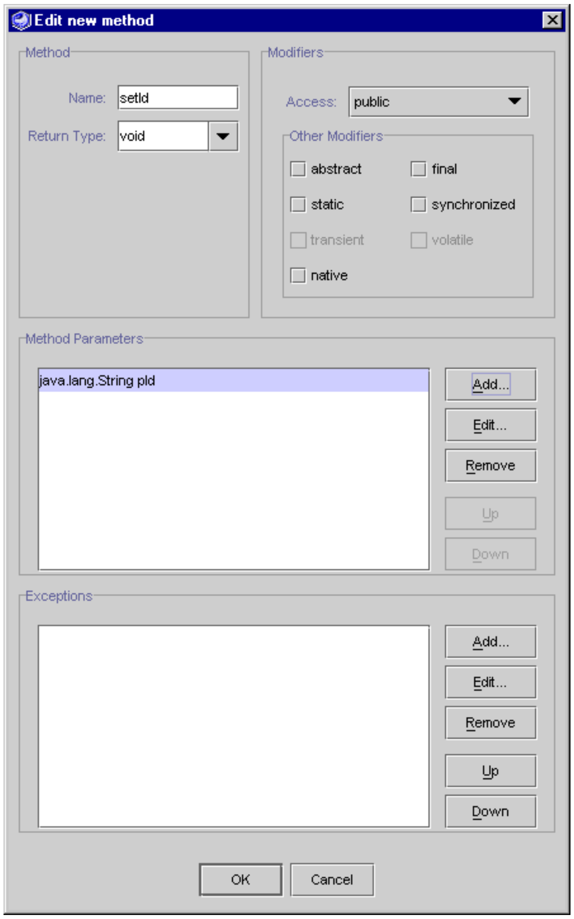
To overload the `setId` method and `setPrice` methods:

1. **Right-click the `Methods` node (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: **`pId`**
5. **Click OK.**

   The New Method dialog box should look like this:


Edit new method

Method

Name: setId

Return Type: void

Modifiers

Access: public

Other Modifiers

- [ ] abstract
- [ ] final
- [ ] static
- [ ] synchronized
- [ ] transient
- [ ] volatile
- [ ] native

Method Parameters

java.lang.String pId

Exceptions

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

7. **Add code to this new method, as follows:**

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

Now do the same with the `setPrice` method.

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

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

   Method Parameter –
   Type: **`java.lang.String`**
   Name: **`pPrice`**

9. **Add the following code to this new method:**

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

10. **Select the** `CartLineItem` **bean ( ) (not the class) and choose Build > Compile.**

    If the bean compiles without errors, the red "need to compile" badge is removed from the bean's node and you are ready to create the `Cart` bean. If not, check your typing and recompile.

Creating 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` bean.

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

2. **Name the bean** `Cart`**.**

3. **Right-click the** `Cart` **bean's** `Bean Patterns` **node and choose New > Property.**

4. **With the New Property Patterns dialog box, create a property with the following values:**

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

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

   You must change the access of the field from private to public.

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

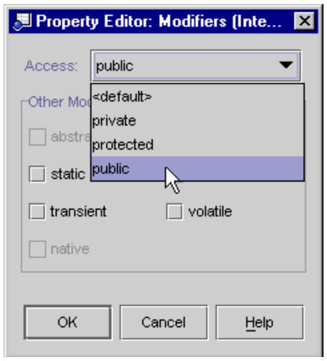
7. **Open its Properties window and click the Modifiers field.**

Note – If the Properties window is not already open (it's usually under the Explorer window), then right-click `lineItems` and choose Properties from the contextual menu. If you want the Properties window to stay up, choose Window > Windows > Browsing > Properties Window.

---

8. **Click the browser button (…) to display the Modifiers dialog box.**

9. **Choose `Public` from the Access list.**



10. **Click OK to accept the change.**

    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.

11. **Open the Constructors node of the Cart bean and double-click the Cart() constructor.**

    This action takes you to the `Cart()` constructor in the Source Editor.

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

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

13. **Right-click the Cart `Methods` node, choose New Method..., and enter the following values for this method:**

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

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

14. **Add the following code in the Source Editor to the `findLineItem` method:**

```
public int findLineItem(int pID) {
    System.out.println("Entering Cart.findLineItem()");
    //Return the element number of the item in the cartItems
    //as specified by the passed ID.
    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;
}
```

15. **Create the `removeLineItem` method with the following values:**

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

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

16. **Add the following code in the Source Editor to the `removeLineItem` method:**

```
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()");
}
```

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

    The bean should compile without errors. Now create the `ShopCart` JSP page.

Creating the Shopping Cart Page

Now 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.

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 Source Editor.

   Develop this page by performing the following tasks:

   1. "Adding Code to Add or Remove an Item From the Shopping Cart Table" on page 59.

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

   3. "Adding the Buttons to the Page" on page 62.

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

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

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

As with the `ProductList` JSP page, 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.

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

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

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

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

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

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

4. **Below the heading, use the `usebean` tag to tell the JSP page to use the `Cart` bean.**

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

This code instantiates a Cart object and places it on the session.

Now 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. Add code that gets the `cdID`, `cdTitle`, and `cdPrice` objects and adds them to the `myCart` object.

5. **Begin the code by getting the current operation and defining what happens when a user clicks “Add.”**

```
<%
    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);
        }
```

Next, 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. **Use the following code to specify what happens when the user clicks “Delete.”:**

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

Finally, 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` JSP page (which you’ll create soon). 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. Use the following code:

```
    //If the last item is removed from the cart...
    if (((Vector)session.getAttribute("myLineItems")).size()
        == 0) {
        //End scriptlet temporarily so that you can use the JSP
        //"forward" tag to forward to the EmptyCart page.
        %>
        <jsp:forward page="EmptyCart.jsp" />
        //Resume the scriptlet.
        <%
        }
        %>
```

Using Presentation Tags to Populate the Cart Table

Next, use the `iterator` and `field` tags to iterate through the passed data, much as you did in "Iterating Through the Data With Presentation Tags" on page 45.

1. Start a table for the purchase candidate data by creating 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:

```
<pr:iterator results="myLineItems" >
    <TR>
        <TD><pr:field name="id"/></TD>
        <TD><pr:field name="cdtitle"/></TD>
        <TD><pr:field name="price"/></TD>
```

The `field` tags in the code above retrieve the value from the named fields in the current row of the results.

3. **Create a Delete button for each row, as in "Adding the Buttons to the Page" on page 62.**

```
<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>
```

Adding the Buttons to the Page

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

- **Add the following code to the** `ShopCart` **JSP page.**

```
<p>
//Create the Resume Shopping button.
<form method=get action="ProductList.jsp">
<input type=submit value="Resume Shopping">
</form>
//Create the Place Order button.
<form method=get action="PlaceOrder.jsp">
<input type=submit value="Place Order">
</form>
//Create the Cancel Order button.
<form method=get action="CancelOrder.jsp">
<input type=submit value="Cancel Order">
</form>
</p>
//End the page.
</body>
</html>
```

Test Executing the Shopping Cart Page

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

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

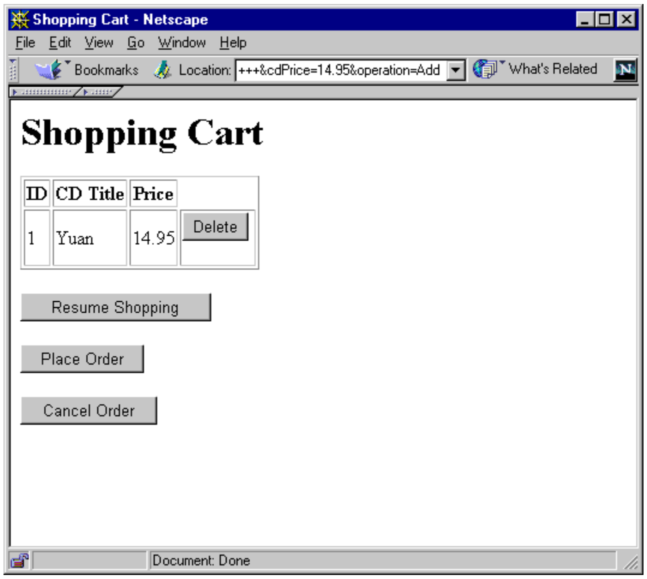
   Everything should compile successfully.

2. **Right-click the `ProductList` JSP page and choose Execute (Force Reload).**

   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 similar to this.



Shopping Cart

ID	CD Title	Price	
1	Yuan	14.95	Delete

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

5. **Terminate the execution (Right-click the process in the Execution window and choose Terminate Process).**

6. **Return to the Editing work space 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

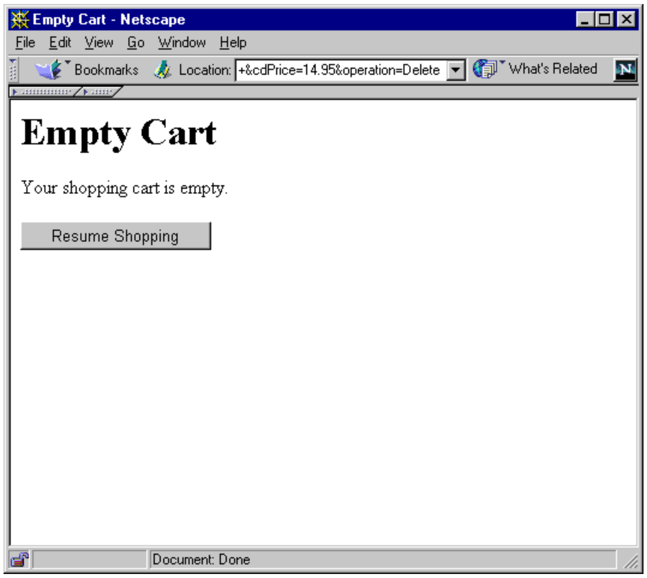
Now create a JSP page that displays when a user empties the cart and two more that display when the user clicks the Place Order and Cancel Order buttons, respectively.

In this section, perform the following tasks:

- "Creating the Empty Cart Page" which follows
- "Creating the Place Order Page" on page 66
- "Creating the Cancel Order Page" on page 68

Creating the Empty Cart Page

When an iterator tag finds an empty vector, it throws an exception (rather than creating an empty table). You have dealt with this by testing for this case (Step 7 on in "Adding 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.



Empty Cart

Your shopping cart is empty.

FIGURE 3-3 Empty Cart Page

To create the Empty Cart page:

1. **Right-click the `CDShopCart` web module and choose New > JSP & Servlet > JSP (HTML).**

2. **Name it `EmptyCart` and click Finish.**

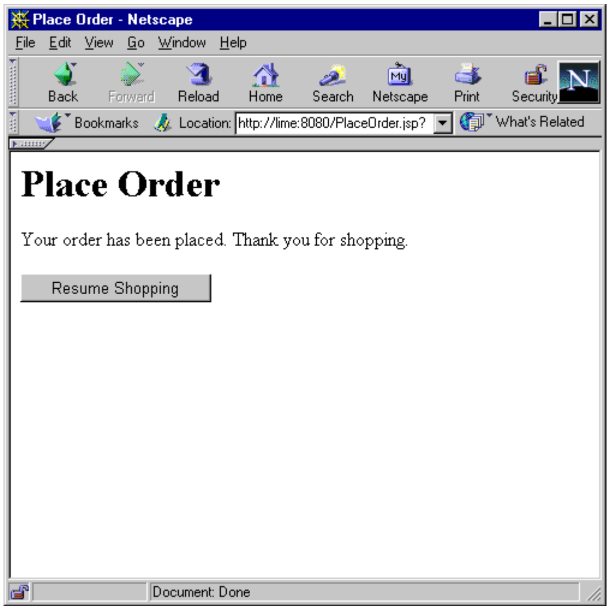
   The EmptyCart JSP page appears in the explorer, and its source code in the Source Editor.

3. **Change the page title to "Place Order" and add code as follows:**

```
<%@page contentType="text/html"%>
<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>
    </P>
    </body>
</html>
```

4. **Save the `EmptyCart` page with File > Save.**

Creating the Place Order Page


Place Order - Netscape
File Edit View Go Window Help
Back Forward Reload Home Search Netscape Print Security
Bookmarks Location: http://lime:8080/PlaceOrder.jsp? What's Related
Resume Shopping
Document: Done

Place Order

Your order has been placed. Thank you for shopping.

FIGURE 3-4 Place Order Page

The Place Order and the Cancel Order page are very simple pages, and present only one of many ways you can implement 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 implementation.

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

To create the Place Order page:

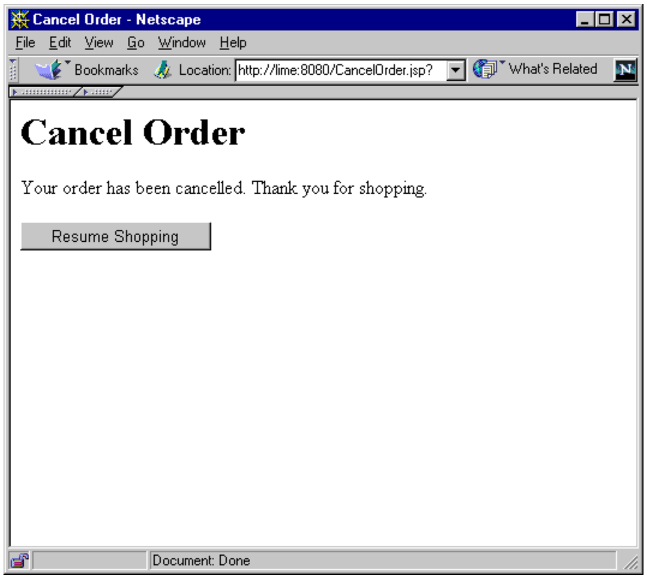
1. **Right-click the `CDShopCart` web module and choose New > JSP & Servlet > JSP (HTML).**
2. **Name it `PlaceOrder` and click Finish.**

3. **Change the page title to** `Place Order` **and add code as follows:**

```
<%@page contentType="text/html"%>
<html>
<head><title>Place Order</title></head>
<body>
    //Add a heading.
    <H1> Place Order </H1>
    //Invalidate the session.
    <%
    session.invalidate();
    %>
    //Add a message.
    Your order has been placed. Thank you for shopping.
    //Add a space.
    <P>
    //Add an Add button.
    <FORM method=get action="ProductList.jsp">
    <INPUT type=submit value="Resume Shopping">
    </FORM>
    </P>
</body>
</html>
```

4. **Save the** `PlaceOrder` **page with File > Save.**

Creating the Cancel Order Page


Cancel Order - Netscape
File Edit View Go Window Help
Bookmarks Location: http://lime:8080/CancelOrder.jsp? What's Related
Resume Shopping
Document: Done

Cancel Order

Your order has been cancelled. Thank you for shopping.

FIGURE 3-5 Cancel Order 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 FIGURE 3-5.

To create the Cancel Order page:

1. **Right-click the `CDShopCart` web module and choose New > JSP & Servlet > JSP (HTML).**

2. **Name the new JSP page `CancelOrder` and click Finish.**

3. **Change the page title to "Cancel Order" and add code similar to the Place Order page, as follows:**

```
<%@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>
    </P>
</body>
</html>
```

4. **Save the `CancelOrder` page with File > Save.**

Test Executing the Three Message Pages

As with the Shopping Cart page, you test execute 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.

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 (Force Reload).**

   Alternatively, you can right-click the `WEB-INF` node and choose Execute (Force Reload).

   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 the message in the Execution window and choose Terminate Process.**

    This action terminates the Tomcat Web Server 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, 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 Forte for Java Programming Series: *Programming Persistence* provides details of what portion of this technology is offered in Forte for Java, Community 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 the classes 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 Persistence mechanism. The application can use them for such things as values derived from persistent fields, 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 Persistence 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). The Persistence Manager is the factory for creating transaction objects and query objects. The `query` class provides a set of database query methods.

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) occurs 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 allows a user to create an order, but when the user clicks the Place Order button, it doesn't actually save the order to the database; it only displays a message about the order being placed. In this chapter, you add components that save the order to the database.

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 75, you use the tables you installed in the database in Chapter 1.

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 "Enhancing the Persistence-Capable Classes" on page 85, you package them in a JAR file within the `WEB-INF/lib` subdirectory of 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 89, 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 the CheckOutBean" on page 97.

5. Test run the entire application.

   You do this in "Test Running the New CDShopCart Application" on page 98.

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, generate the persistence-capable classes from the three tables you created for the CDShopCart application. Then compile the package and enhance it.

In this section, perform the following procedures:

1. "Capturing the Database Schema," which follows
2. "Generating the Persistence-Capable Classes" on page 80
3. "Enhancing the Persistence-Capable Classes" on page 85

Capturing the Database Schema

The captured schema must be stored in a package. Create this package outside the `CDShopCart` web module within a mounted Filesystem. Later, enhance the classes by packaging them in a JAR file that you create within the `CDShopCart` web module.

To capture the Tutorial Database schema:

1. **Choose File > Mount Filesystem.**

   This action displays the Mount Filesystem dialog box.

2. **Navigate to the location you want to create the file system.**

   In this example, it is directly under the C: disk, but it can be anywhere.

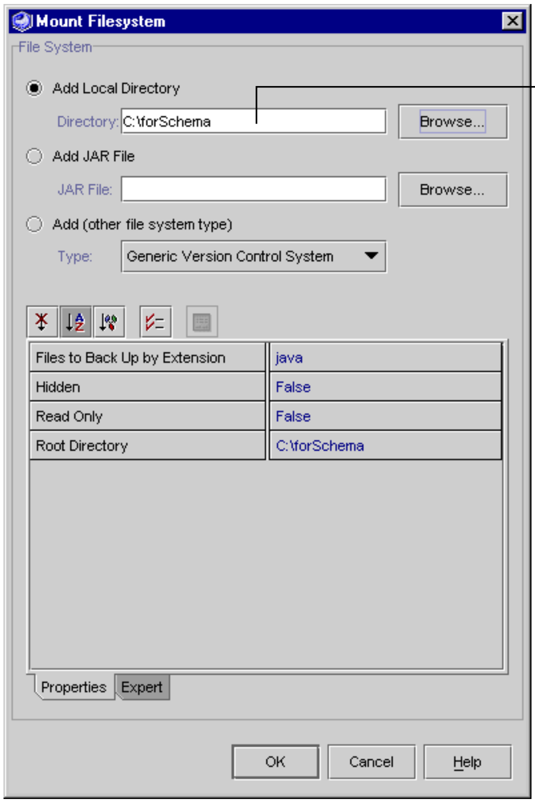
3. **Click the New Folder button.**

   This creates a folder named New Folder in the folder display.

4. **Select New Folder in the folder display and rename it `forSchema`.**

5. **Click OK.**

   The Mount Filesystem dialog box should look something like this:



Directory: C:\forSchema

JAR File:

Type: Generic Version Control System

Files to Back Up by Extension	java
Hidden	False
Read Only	False
Root Directory	C:\forSchema

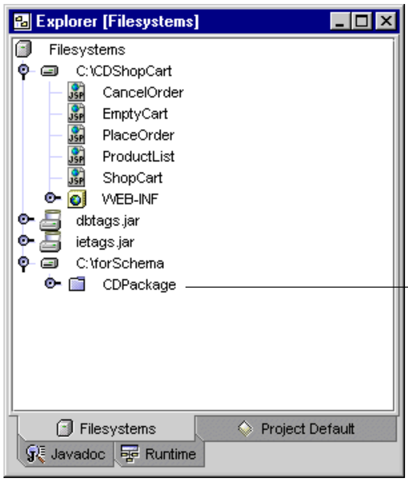
6. **Click OK.**

   The forSchema folder appears in the Explorer.

7. **Right-click the** `forSchema` **node and choose New Package from the contextual menu.**

8. **Name the package `CDPackage` and click OK.**

   The new package appears in the Explorer.



9. **Right-click on the `CDPackage` package and choose New > Databases > Database Schema.**

   The New Object Name pane of the New From Template wizard appears.

10. **Name the new schema `cdschema` and click Next.**

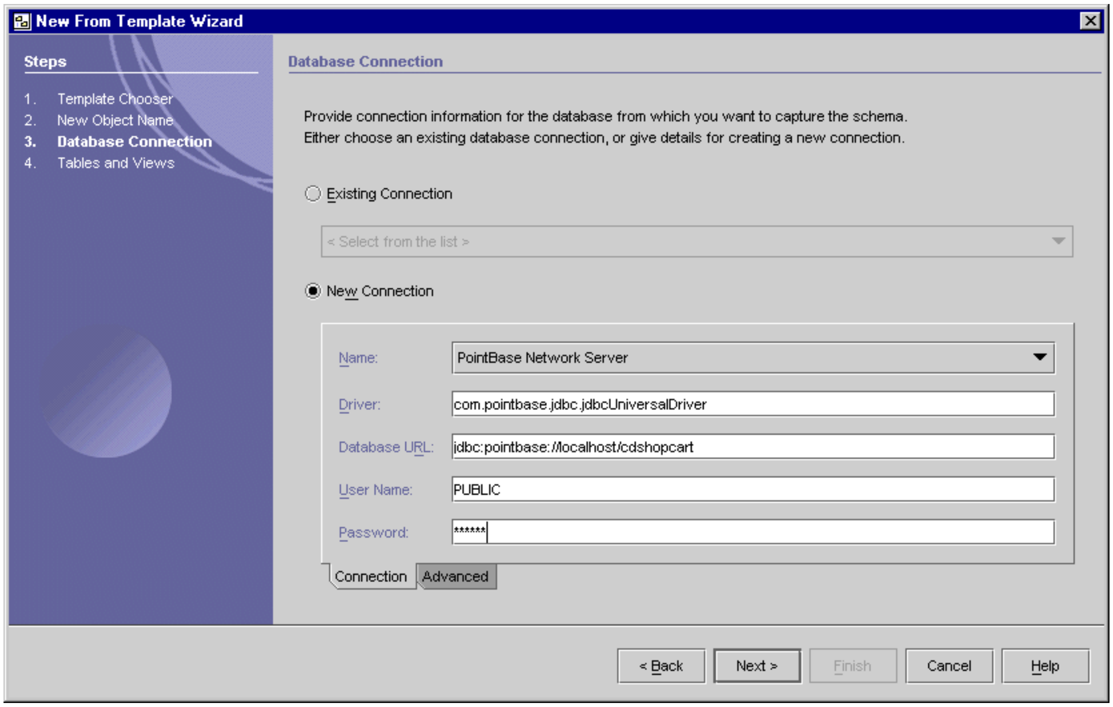
    A pane for connection information appears.

11. **Enable the New Connection option and enter the following values:**

    Name: **`PointBase Network Server`** (use the drop-down list)
    Driver: **`com.pointbase.jdbc.jdbcUniversalDriver`**
    Database URL: **`jdbc:pointbase://localhost/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 “Using the JDBC connection Tag to Connect to the Database” on page 43.

The page should look like this:



Provide connection information for the database from which you want to capture the schema.
Either choose an existing database connection, or give details for creating a new connection.

Existing Connection

< Select from the list >

New Connection

Name: PointBase Network Server

Driver: com.pointbase.jdbc.jdbcUniversalDriver

Database URL: jdbc:pointbase://localhost/cdshopcart

User Name: PUBLIC

Password: ******

12. **Click Next.**

    The next pane is used to choose which tables you want to capture. You'll want to capture all the tables in the `cdshopcart` schema.

13. **Click the Add All button to select all the tables for capture.**

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

The page should look as follows:

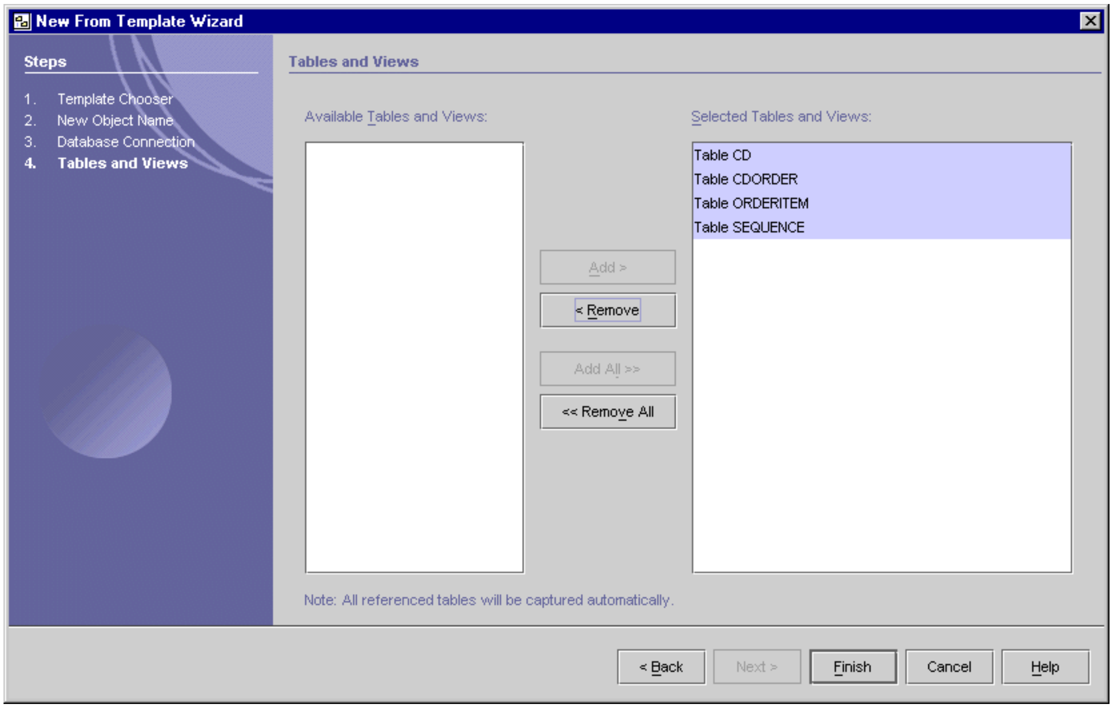


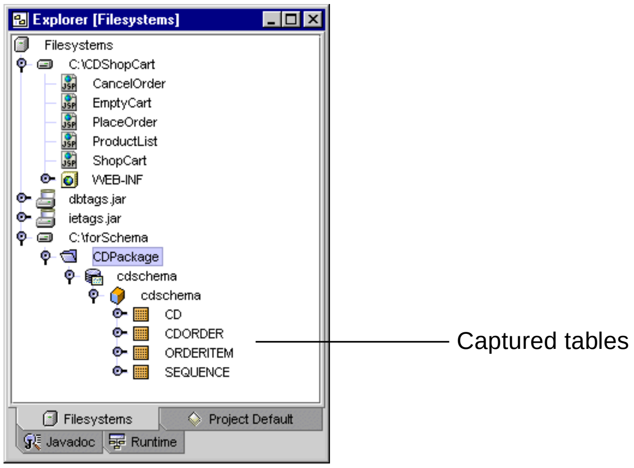
Table CD
Table CDORDER
Table ORDERITEM
Table SEQUENCE

Note: All referenced tables will be captured automatically.

14. **Click Finish.**

    A progress window appears and when it closes, the new database schema is displayed in the `CDPackage` package.

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



Now generate the persistence-capable classes.

Generating the Persistence-Capable Classes

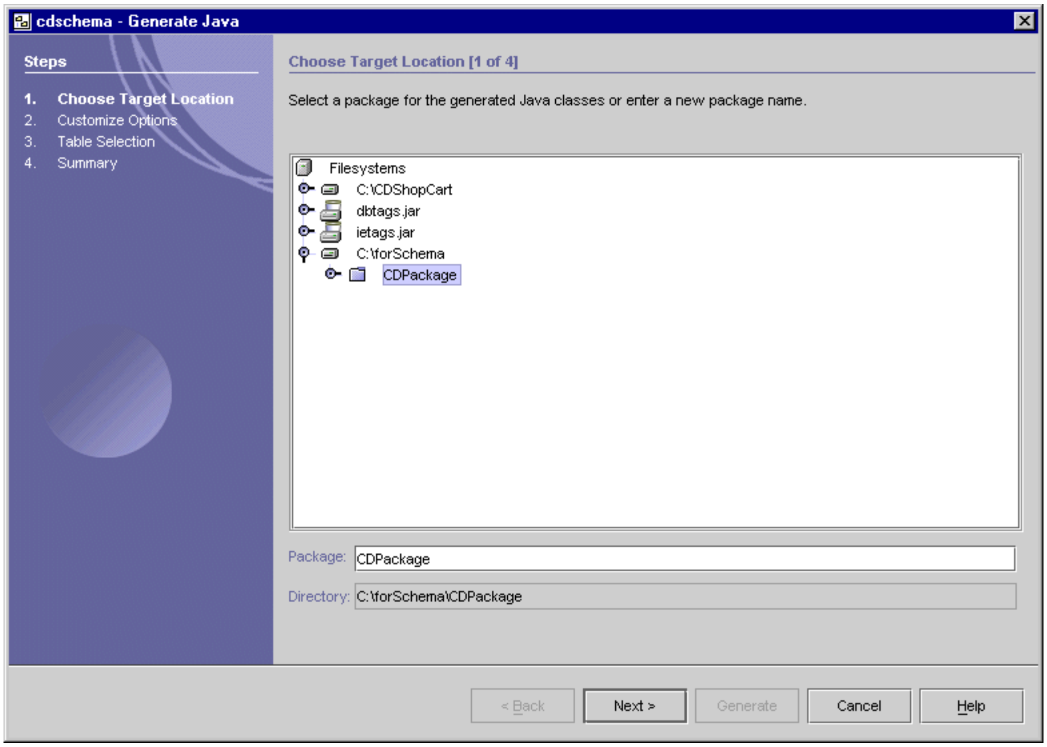
Generate the persistence-capable classes from some of the tables in the database schema that you just captured.

1. **Right-click the `cdschema` node ( ) just under the `CDPackage` package, and choose Generate Java from the contextual menu.**

   This displays the Choose Target Location page of the Generate Java wizard.

2. **Expand the `forSchema` Filesystem and select the `CDPackage` package.**

   The wizard should look like this:



Select a package for the generated Java classes or enter a new package name.

Package: CDPackage

Directory: C:\forSchema\CDPackage

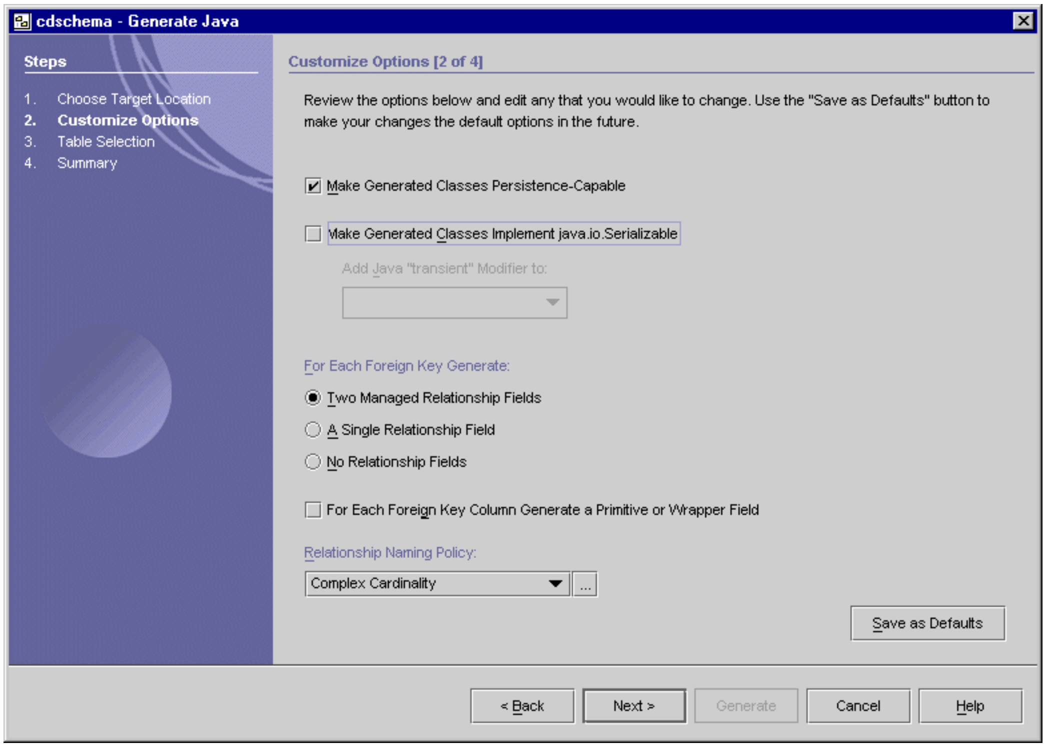
3. **Click Next.**

   The Customize Options pane of the Generate Java wizard is displayed.

4. **Set the following options:**

   Make Generated Classes Persistence-Capable: **`enabled`**
   Make Generated Classes Implement java.io.Serializable: **`disabled`**
   For Each Foreign Key Generate: **`Two Managed Relationship Fields`**
   For Each Foreign Key Column Generate a Primitive or Wrapper Field: **`disabled`**
   Relationship Naming Policy: **`Complex Cardinality`**

   The wizard page should look like this:



cdschema - Generate Java

Steps

1. Choose Target Location
2. Customize Options
3. Table Selection
4. Summary

Customize Options [2 of 4]

Review the options below and edit any that you would like to change. Use the "Save as Defaults" button to make your changes the default options in the future.

[✓] Make Generated Classes Persistence-Capable

[ ] Make Generated Classes Implement java.io.Serializable

Add Java "transient" Modifier to:

For Each Foreign Key Generate:

(●) Two Managed Relationship Fields

( ) A Single Relationship Field

( ) No Relationship Fields

[ ] For Each Foreign Key Column Generate a Primitive or Wrapper Field

Relationship Naming Policy:

Complex Cardinality

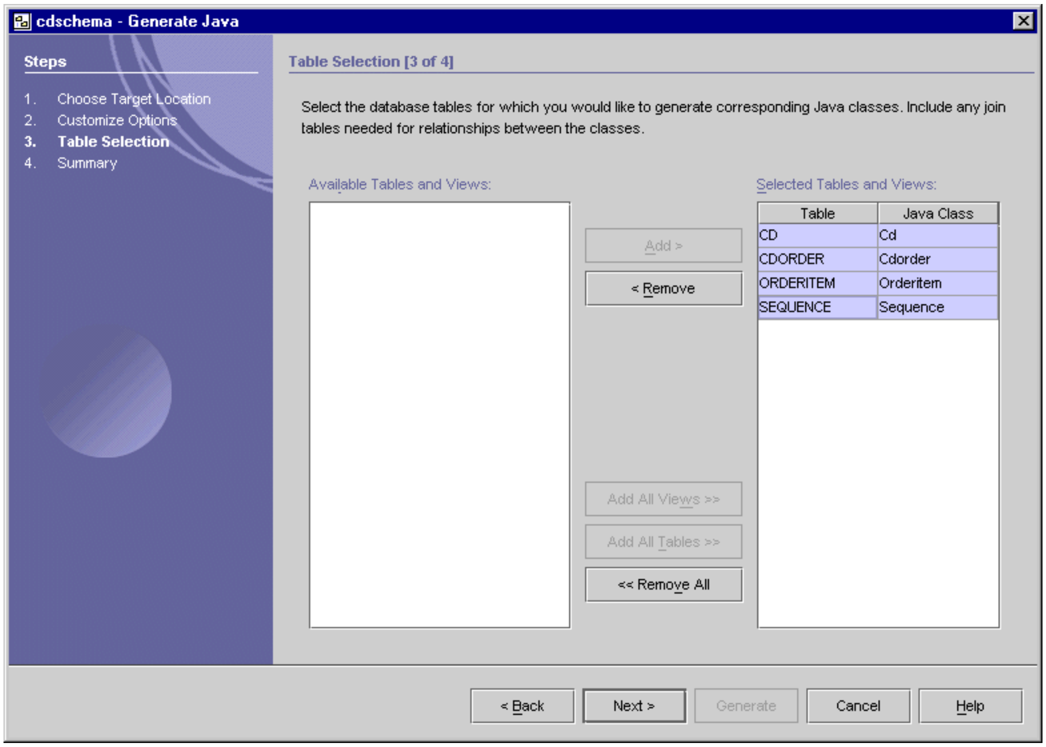
5. **Click Next.**

   The Table Selection pane of the Generate Java wizard is displayed.

6. **Click the Add All Tables button 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 should look like this:



Select the database tables for which you would like to generate corresponding Java classes. Include any join tables needed for relationships between the classes.

Available Tables and Views:

Selected Tables and Views:

Table	Java Class
CD	Cd
CDORDER	Cdorder
ORDERITEM	Orderitem
SEQUENCE	Sequence

7. **Click on `CdOrder` label under Java Class, change the name to `Order`, and press Enter to confirm your change.**

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

   The Selected tables and views section should look like this:

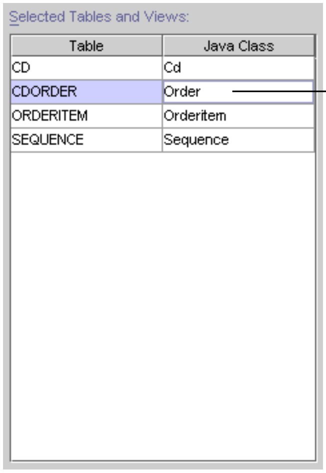
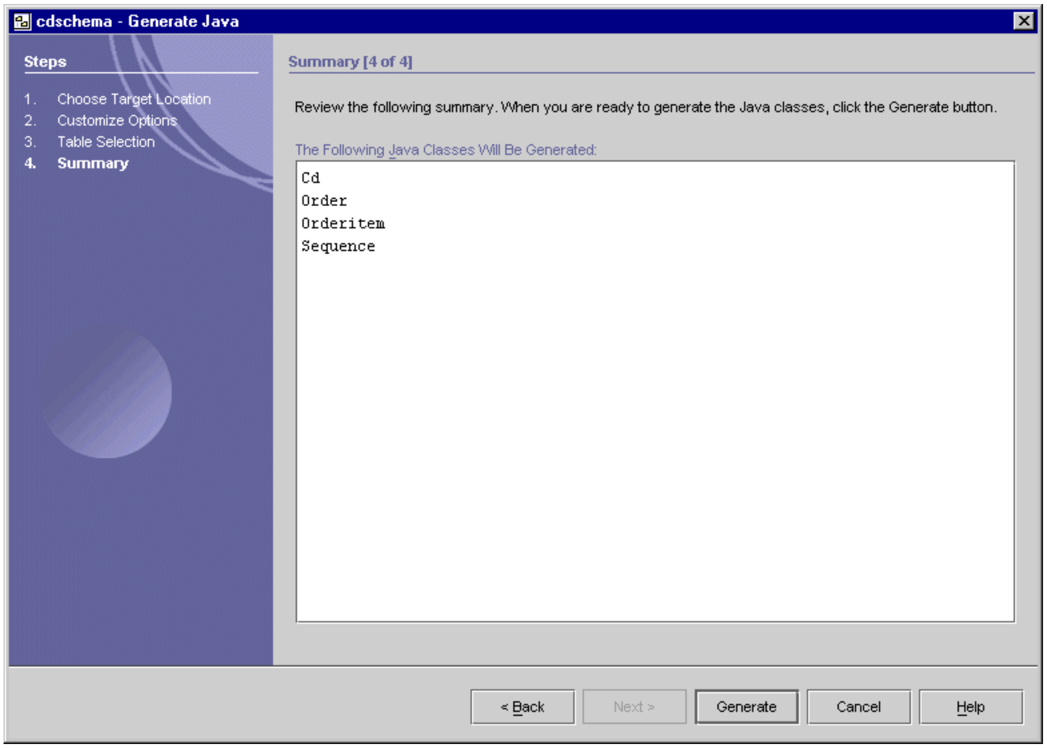

Selected Tables and Views:

Table	Java Class
CD	Cd
CDORDER	Order
ORDERITEM	Orderitem
SEQUENCE	Sequence

New label text of CdOrder.

8. **Click Next.**

   The wizard displays a summary pane of the classes it will generate.



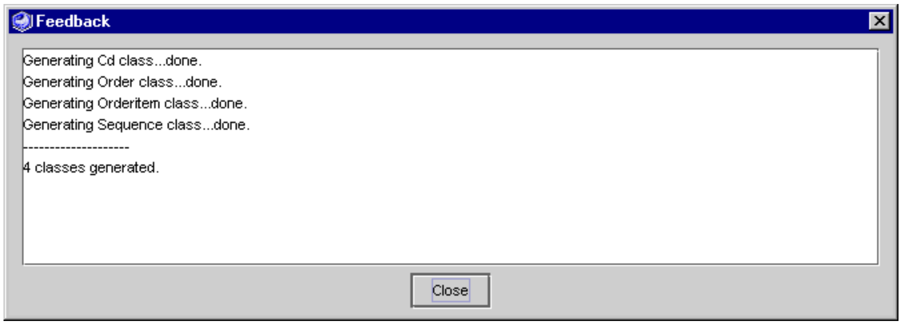
Review the following summary. When you are ready to generate the Java classes, click the Generate button.

The Following Java Classes Will Be Generated:

```
Cd
Order
Orderitem
Sequence
```

9. Click Generate.

The IDE displays a Feedback window showing which classes were generated.

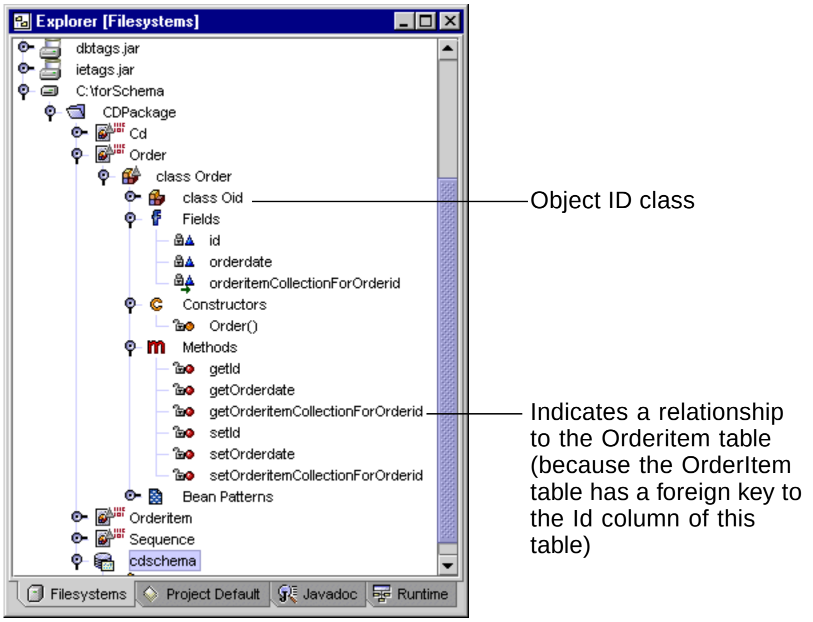


```
Generating Cd class...done.
Generating Order class...done.
Generating Orderitem class...done.
Generating Sequence class...done.
-------------------
4 classes generated.
```

10. Click Close to close the window.

11. In the Explorer, open one of the newly generated classes.

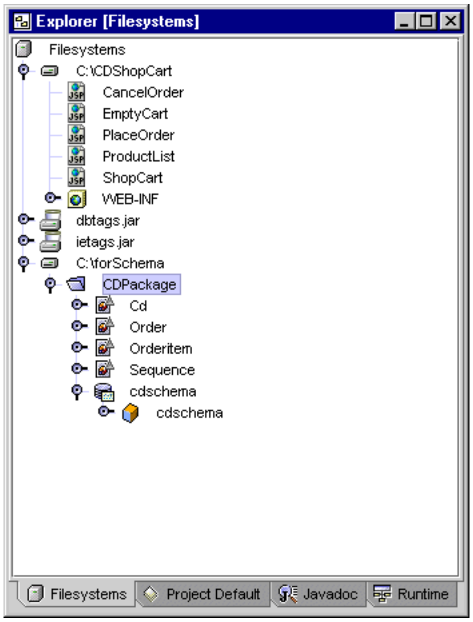
Observe that the fields were generated from the table's columns and that each has `get` and `set` methods.



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.

12. **Compile `CDPackage` by right-clicking it and choosing Compile.**

    Notice that the persistence-capable classes no longer have the red badge of uncompiled classes:



    Now enhance the persistence-capable classes.

Enhancing 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 enable 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.

Note – There are two methods of packaging persistence-capable classes within a web module. Which method you use depends on whether the persistence-capable classes are subject to frequent changes or not. Please refer to the JavaServer Pages™/Servlet subsection of the Modules Notes section of the Release Notes.

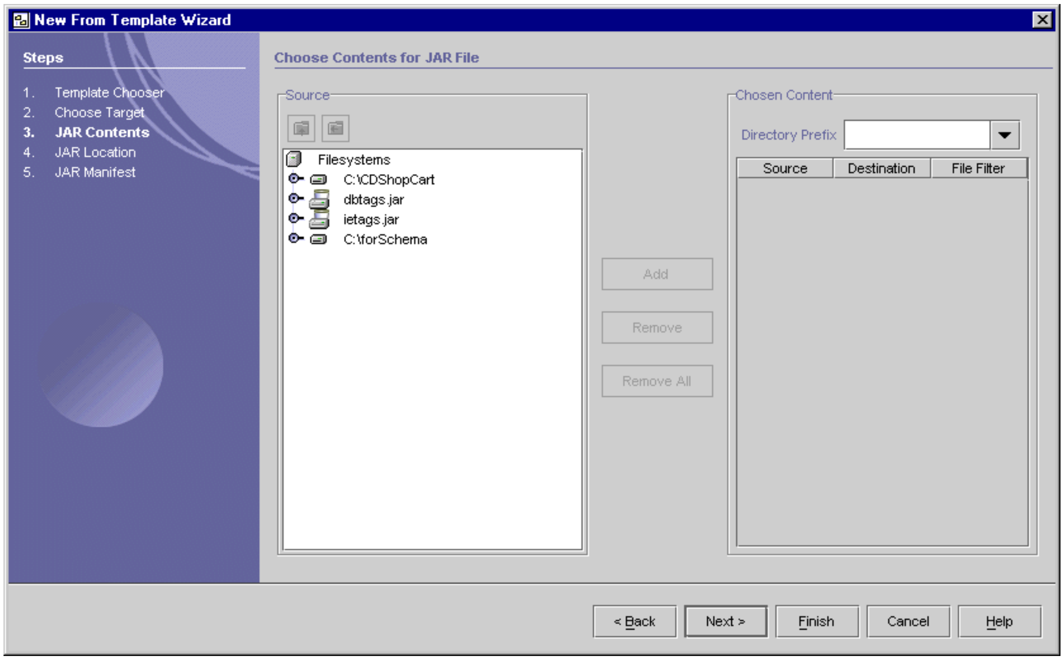
To enhance the persistence-capable classes:

1. **Right-click the `lib` node under the CDShopCart's `WEB-INF` node and choose New > JAR Packager > Jar Contents.**

   The New From Template wizard displays the New Object Name pane.

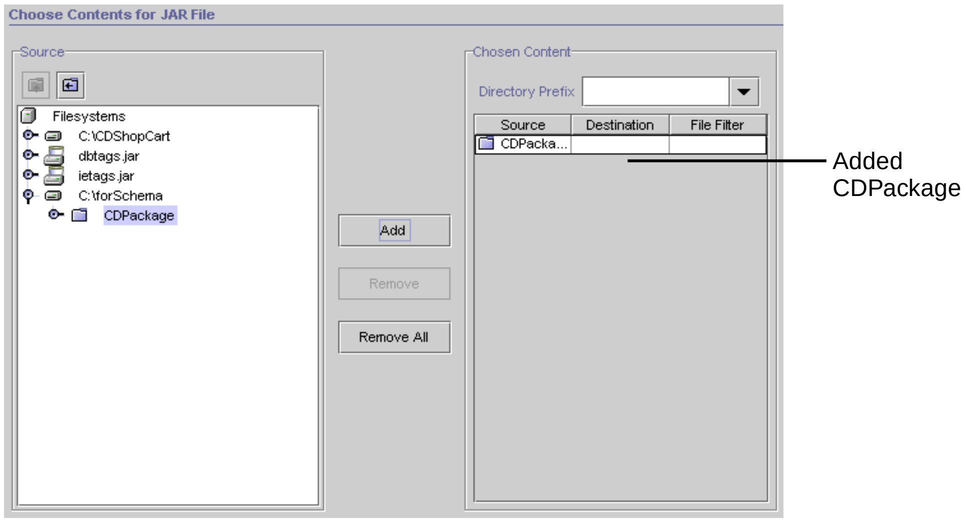
2. **Type `CDclasses` for the name and click Next.**

   The JAR Contents pane is displayed, which allows you to choose the contents of the new JAR file or modify the contents of an existing JAR file, as shown:



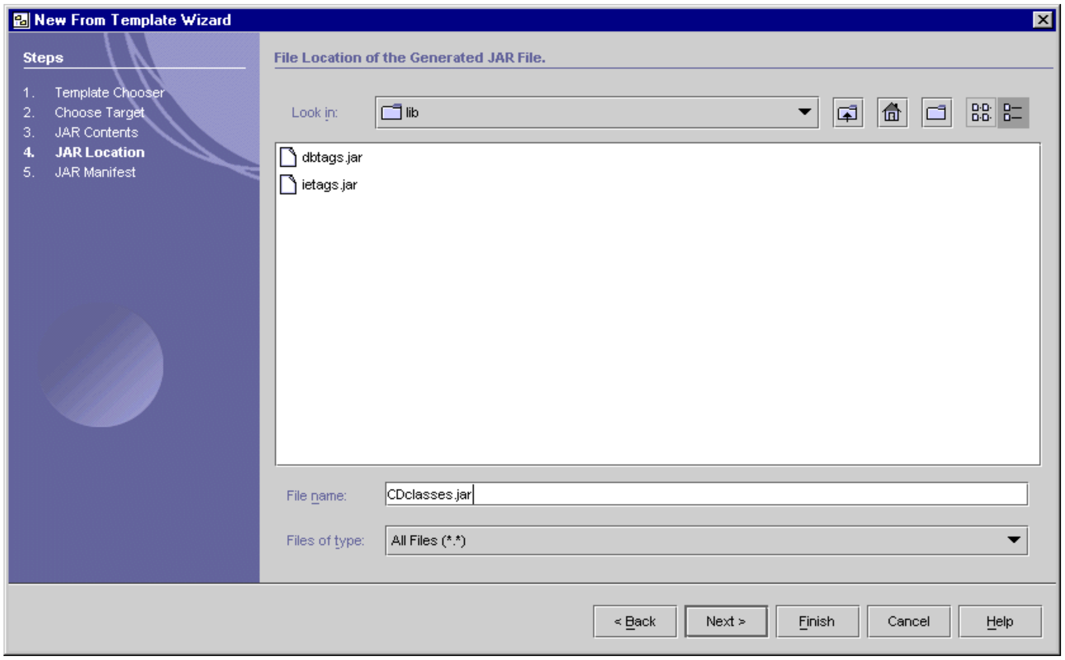
3. **Find the `CDPackage` package, select it, and click the Add button.**

   The CDPackage appears under Source in the Chosen content pane.



4. **Click Next.**

   A wizard pane is displayed for choosing the target location of the JAR file. The `lib` folder is in the "Look in" field and `CDclasses.jar` is in the File name field, as shown.

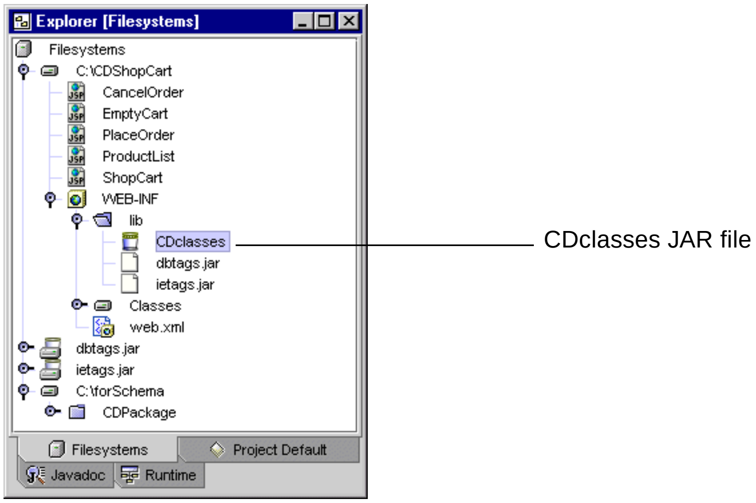


5. **Click Next.**

   The JAR manifest pane is displayed.

6. **Click Finish to generate the JAR package.**

   The `CDPackage` JAR file appears in the Explorer in the `WEB-INF/lib` folder, as shown.



Moving a Filesystem in the CLASSPATH

Now make sure the old persistence-capable (but unenhanced) classes in `CDPackage` are a lower file system than the `CDShopCart` file system in your CLASSPATH. You must do this so that the compiler uses the enhanced classes instead of the unenhanced ones.

1. **Right-click `Filesystems` in the Explorer and choose Customize.**

   This displays the Customizer Dialog box. Find both the `CDShopCart` and `forSchema` file systems in the list.

2. **If the `forSchema` file system is above the `CDShopCart` file system, right-click the `forSchema` file system and choose Move Down.**

   If the `forSchema` file system is below the `CDShopCart` file system (and the `CDShopCart: /WEB-INF/classes` file system), you don't need to do anything.

3. **Close the Customizer Dialog box.**

Now create the `CheckOutBean` bean that encapsulates the Transparent Persistence.

Creating the Persistence-Aware Bean

Now create the component that encapsulates the Transparent Persistence functionality, the `CheckOutBean` bean. The `CheckOutBean` bean creates a Persistence Manager Factory to specify the properties of the database connection. It then creates a Persistence Manager. Use the Persistence Manager's methods to create `transaction` objects and `query` objects. Use the `query` class's methods to perform the database query functions.

1. **Open the `WEB-INF` folder of the `CDShopCart` web module.**
2. **Right-click the `Classes` folder and choose New > Beans > Bean.**
3. **Type `CheckOutBean` in the Name field and click Finish.**

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

   Now add the following modifications to the `CheckOutBean` bean's code:

   1. Initialize the Persistence Manager Factory and a Persistence Manager.
   2. Fetch a CD based on an ID.
   3. Add an order and line items for each item in the cart.
   4. Get a sequence number for the next order.

Initializing the Persistence Manager Factory and Persistence Manager

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

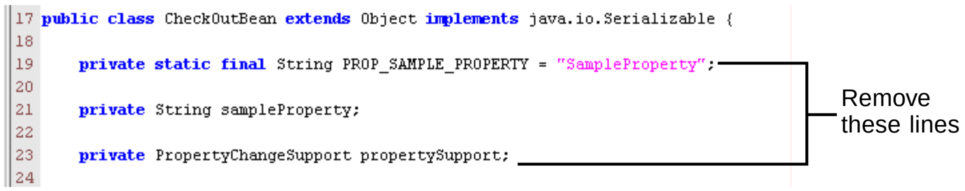
To initialize the Persistence Manager Factory and Persistence Manager:

1. **In the Editor, add three new import statements:**

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

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

2. **Remove the first three statements of the `CheckOutBean` declaration:**



```
public class CheckOutBean extends Object implements java.io.Serializable {

    private static final String PROP_SAMPLE_PROPERTY = "SampleProperty";

    private String sampleProperty;

    private PropertyChangeSupport propertySupport;

```

3. **Replace these lines with the following code.**

```
private PersistenceManager pm;
```

This code declares a Persistence Manager object.

4. **Replace the `CheckOutBean`'s constructor code with the following code to initialize the Persistence Manager Factory and create a Persistence Manager.**

```
public CheckOutBean() {
    // Instantiate a Persistence Manager Factory:
    PersistenceManagerFactory pmf = new
            PersistenceManagerFactoryImpl();
    // Specify the database connection the Persistence Manager
    // is to use:
    pmf.setConnectionUserName("PUBLIC");
    pmf.setConnectionPassword("PUBLIC");
    pmf.setConnectionDriverName(
            "com.pointbase.jdbc.jdbcUniversalDriver");
    pmf.setConnectionURL("jdbc:pointbase://localhost/cdshopcart");
    // Specify the default concurrency control for all
    // Persistence Manager transactions:
    pmf.setOptimistic(false);
    //Create a Persistence Manager:
    this.pm = pmf.getPersistenceManager();
}
```

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

---

5. **Save your work so far with File > Save.**

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.

Fetching a CD Based on an ID

Now 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.

1. **Delete all of the remaining default code in the `CheckOutBean`.**
2. **In the Explorer, right-click the `CheckOutBean`'s `Methods` node 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();
    //Bind the result class (Cd) to query instance:
    q.setClass(Cd.class);
    //Define the input collection for the query:
    q.setCandidates(pm.getExtent(Cd.class, false));
    //Bind the query filter to the query instance:
    q.setFilter("id == CDid");
    //Define a parameter for the placeholder:
    String param = "Long CDid";
    //Bind the parameter to the query statement:
    q.declareParameters(param);
    //Execute the query and return the filtered collection:
    Collection result = (Collection)q.execute(new Long(id));
    //Iterate through the result to find the specified CD:
    Iterator i = result.iterator();
    Cd theCd = null;
    if (i.hasNext()) {
        theCd = (Cd)i.next();
    }
    return theCd;
}
```

6. **Save your work.**

The query in this code includes the three statements required in any query, namely:

- 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

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.

Adding an Order and Line Items for Each Item in the Cart

Create a `checkout` method that will 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.

Create the `checkout` method:

1. **Create a new method for the `CheckOutBean` with the following values:**

   Method –
   Name: **`checkout`**
   Return type: **`int`**
   Access: **`public`**

   Parameter –
   Type: **`Cart`**
   Name: **`myCart`**

2. **Type the following code into the body of the `checkout` method:**

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

   This code uses a Persistence Manager method to create the transaction and then starts the transaction.

3. **Following the previous line, add the following code to create the order:**

```
//Get the next Order sequence number.
int ordNum = this.getSequenceNumber("CDORDER", 1);
//Create a new order.
Order ord = new Order();
//Set the primary key for the order.
ord.setId(ordNum);
//Set the current date.
ord.setOrderdate(new Date());
//Tell the Persistence Manager to mark for database update.
pm.makePersistent(ord);
```

4. **Under the previous lines, add code that adds a line item to the order for each item in the cart:**

```
//Initialize the Line item number.
int itemNum = 1;
//Create a new hash list to store all line items.
HashSet itemList = new HashSet();
Iterator i = myCart.lineItems.iterator();
while (i.hasNext()) {
    //For the next item in cart
    CartLineItem c = (CartLineItem)i.next();
    //create a new line item
    Orderitem item = new Orderitem();
    //Set the primary key
    item.setOrderid(ordNum);
    item.setLineitemid(itemNum++);
    item.setCdOfProductid(getCd(c.getId()));
    //Add to the collection
    itemList.add(item);
    //Tell the PM to mark for a database update
    pm.makePersistent(item);
}
```

5. **Following this code, add code to commit this transaction and return the order number:**

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

6. **Save your work.**

Getting a Sequence Number for the Next Order

In this section, you create the `getSequenceNumber` method, which is the final method of the `CheckOutBean`.

To create the `getSequenceNumber` method:

1. **Create a new method for the `CheckOutBean` with the following values:**

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

   Parameter –
   Type: **`java.lang.String`**
   Name: **`tableName`**

   Parameter –
   Type: **`int`**
   Name: **`amount`**

2. **In the Editor, add the following code to the `getSequenceNumber` method to initialize the key:**

```
int key = 0;
```

3. **Following this line, add 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:
Query sequenceQuery = pm.newQuery();
// Bind the result class (Sequence) to the query instance:
sequenceQuery.setClass(Sequence.class);
// Define the input collection. The filter, tablename, is
// the field, and name is the parameter name.
sequenceQuery.setCandidates(pm.getExtent(Sequence.class, false));
sequenceQuery.setFilter("tablename == name");
// Define the parameter:
String param = "String name";
// Bind the parameter to the query statement:
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 the caller:
return key;
```

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

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

Modifying the PlaceOrder Page to Call the `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. Page directive to import the `Cart` and `CheckOutBean` beans.
2. `usebean` tags to allow the `Cart` and `CheckOutBean` beans to be used by the page.
3. Declare an order number.
4. Set the order number to the one returned by the `CheckOutBean`.
5. Modify the displayed message to include the order number.

To modify the PlaceOrder JSP page:

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

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

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

```
//Use the Cart bean.
<jsp:useBean id="myCart" scope="session" class="Cart" />
//Use the CheckOutBean bean.
<jsp:useBean id="checker" scope="session" class="CheckOutBean" />
```

3. **Under these statements, add the following code:**

```
<%! int ordNum; %>

<%
ordNum = checker.checkout(myCart);
session.invalidate();
%>
```

   The second statement sets the `ordNum` to the order number returned by the `checkout` method.

4. **And finally, modify the displayed message to use the `ordNum`, as follows:**

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

Thank you for shopping.
```

5. **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 Executing the Three Message Pages" on page 69), 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 All.**

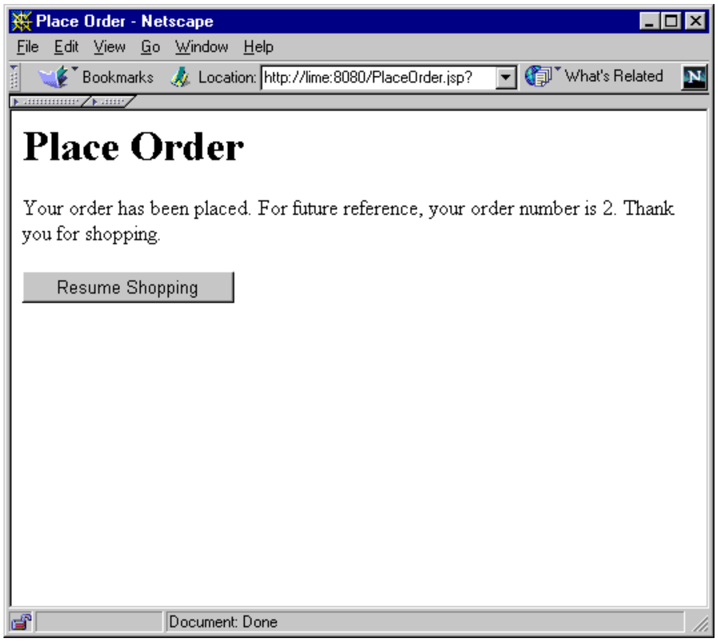
   Everything should compile successfully.

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. **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:**



Place Order

Your order has been placed. For future reference, your order number is 2. Thank you for shopping.

Congratulations! You have completed the CDShopCart tutorial!

Index