Business Component Development with EJB Technology, Java EE 5

Activity Guide

SL-351-EE5 REV D.2

D61838GC10
Edition 1.0
D62448

ORACLE

Disclaimer

The information contained in this document is subject to change without notice. If you find any problems in the document, please report them in writing to: Oracle University, 500 Oracle Parkway, Redwood Shores, California 94065 USA. This document is not warranted to be error-free.

Sun Microsystems, Inc. Disclaimer

This training manual may include references to materials, offerings, or products that were previously offered by Sun Microsystems, Inc. Certain materials, offerings, services, or products may no longer be offered or provided.Oracle and its affiliates cannot be held responsible for any such references should they appear in the text provided.

Restricted Rights Notice

If this documentation is delivered to the U.S. Government or anyone using the documentation on behalf of the U.S. Government, the following notice is applicable:

U.S. GOVERNMENT RIGHTS
The U.S. Government's rights to use, modify, reproduce, release, perform, display, or disclose these training materials are restricted by the terms of the applicable Oracle license agreement and/or the applicable U.S. Government contract.

Trademark Notice

Oracle and Java are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

AMD, Opteron, the AMD logo, and the AMD Opteron logo are trademarks or registered trademarks of Advanced Micro Devices. Intel and Intel Xeon are trademarks or registered trademarks of Intel Corporation. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. UNIX is a registered trademark licensed through X/Open Company, Ltd.

This page intentionally left blank.

This page intentionally left blank.

Table of Contents

About This Workbook

Course Goal

Upon completion of this course, you should be able to:

- Examine the Java™ Platform, Enterprise Edition (Java EE)
- Examine a Java EE technology application
- Implement Enterprise JavaBeans™ (EJB™) 3.0 session beans
- Implement Java Persistence API entity classes
- Implement entity composition, association, and inheritance
- Use the Java Persistence API query language
- Develop Java EE technology applications using messaging
- Create message-driven beans
- Implement interceptor classes and methods
- Implement transactions
- Implement exception handling for EJB technology
- Add timer functionality to EJB components
- Implement security for Java EE technology
- Evaluate best practices for EJB technology

Topics Not Covered

This course does not cover the following topics. Many of these topics are covered in other courses offered by Sun Services:

- Object-oriented concepts – Covered in OO-226: *Object-Oriented Analysis and Design for Java™ Technology (UML)*
- Distributed programming concepts and support technologies, such as remote procedure call (RPC), Remote Method Invocation (RMI), Internet Inter-ORB Protocol (IIOP), Common Object Request Broker Architecture (CORBA), Lightweight Directory Access Protocol (LDAP), Java Naming and Directory Interface™ API (JNDI API)

Refer to the Sun Educational Services catalog for specific information and registration.

How Prepared Are You?

To be sure you are prepared to take this course, can you answer yes to the following questions?

- Can you write a Java technology class?
- Can you implement composition, association, and inheritance relationships?
- Can you handle events and exceptions generated in Java technology classes?
- Can you describe the issues associated with transaction management?

Introductions

Now that you have been introduced to the course, introduce yourself to the other students and the instructor, addressing the following items:

- Name
- Company affiliation
- Title, function, and job responsibility
- Experience related to topics presented in this course
- Reasons for enrolling in this course
- Expectations for this course

How to Use Course Materials

To enable you to succeed in this course, these course materials contain a learning module that is composed of the following components:

- Goals – You should be able to accomplish the goals after finishing this course and meeting all of its objectives.
- Objectives – You should be able to accomplish the objectives after completing a portion of instructional content. Objectives support goals and can support other higher-level objectives.
- Lecture – The instructor presents information specific to the objective of the module. This information helps you learn the knowledge and skills necessary to succeed with the activities.
- Activities – The activities take on various forms, such as an exercise, self-check, discussion, and demonstration. Activities help you facilitate the mastery of an objective.
- Visual aids – The instructor might use several visual aids to convey a concept, such as a process, in a visual form. Visual aids commonly contain graphics, animation, and video.

Conventions

The following conventions are used in this course to represent various training elements and alternative learning resources.

Icons

Additional resources – Indicates other references that provide additional information on the topics described in the module.

Discussion – Indicates a small-group or class discussion on the current topic is recommended at this time.

Note – Indicates additional information that can help students but is not crucial to their understanding of the concept being described. Students should be able to understand the concept or complete the task without this information. Examples of notational information include keyword shortcuts and minor system adjustments.

Typographical Conventions

`Courier` is used for the names of commands, files, directories, programming code, and on-screen computer output; for example:

> Use `ls -al` to list all files.
> `system% You have mail`.

`Courier` is also used to indicate programming constructs, such as class names, methods, and keywords; for example:

> The `getServletInfo` method is used to get author information.
> The `java.awt.Dialog` class contains `Dialog` constructor.

`Courier bold` is used for characters and numbers that you type; for example:

> To list the files in this directory, type:
> `# ls`

`Courier bold` is also used for each line of programming code that is referenced in a textual description; for example:

```
1 import java.io.*;
2 import javax.servlet.*;
3 import javax.servlet.http.*;
```

> Notice the `javax.servlet` interface is imported to allow access to its life-cycle methods (Line 2).

`Courier italics` is used for variables and command-line placeholders that are replaced with a real name or value; for example:

> To delete a file, use the `rm` *`filename`* command.

`Courier italic bold` is used to represent variables whose values are to be entered by the student as part of an activity; for example:

> Type **`chmod a+rwx`** ***`filename`*** to grant read, write, and execute rights for *`filename`* to world, group, and users.

Palatino italics is used for book titles, new words or terms, or words that you want to emphasize; for example:

> Read Chapter 6 in the *User's Guide*.
> These are called *class* options.

Additional Conventions

Java programming language examples use the following additional conventions:

- Method names are not followed with parentheses unless a formal or actual parameter list is shown; for example:

  "The `doIt` method..." refers to any method called `doIt`.

  "The `doIt()` method..." refers to a method called `doIt` that takes no arguments.

- Line breaks occur only where there are separations (commas), conjunctions (operators), or white space in the code. Broken code is indented four spaces under the starting code.

- If a command used in the Solaris™ Operating System (Solaris OS) is different from a command used in the Microsoft Windows platform, both commands are shown; for example:

  In the Solaris OS:

  ```
  $cd SERVER_ROOT/BIN
  ```

  In Microsoft Windows:

  ```
  C:\>CD SERVER_ROOT\BIN
  ```

Lab 1

Examining Enterprise JavaBeans (EJB) Applications

Objectives

Upon completion of this lab, you should be able to:

- Complete the review questions

Exercise 1: Completing Review Questions

In this exercise, you answer questions about the material covered in Module 1 of the Student Guide.

Complete the following questions:

1. Which two containers are required to be hosted by a Java EE 5 compliant application server?
2. List four deployment based services provided by an EJB container.
3. Which Java EE API would you use to implement asynchronous communication between Java EE application components?

Lab 2

Introducing the Auction Application

Objectives

Upon completion of this lab, you should be able to:

- Start the NetBeans™ IDE
- Deploy and use the `AuctionApp` application
- Complete the review questions

Exercise 1 – Starting the NetBeans IDE

In this exercise, you will learn to start the NetBeans IDE.

This exercise contains the following sections:

- Task 1 – Starting and Configuring the IDE

Task 1 – Starting and Configuring the IDE

In this task, you start the NetBeans IDE and configure it.

Note – Your home directory `$HOME` should be `/export/home/student` on Solaris OS and `C:\student` on Microsoft Windows.

Complete the following steps:

1. Start the NetBeans IDE.

   Using a terminal window enter the following command:

   `netbeans`

2. Start the application server if you have shut it down.

   Use the `Services` tab of the IDE. Expand the Servers node. Right click `GlassFish V2` and select the `Start` menu option.

Exercise 2 – Running the Auction System

In this exercise, you become familiar with the handling of the auction system.

This exercise contains the following sections:

- Task 1 –Building and Deploying the Auction Application
- Task 2 – Using the Auction System
- Task 3 – Undeploying the Auction System

Task 1 –Building and Deploying the Auction Application

In this task, you configure the application server and JavaDB database server, and deploy the AuctionApp application.

Note – Your home directory `$HOME` should be `/export/home/student` on Solaris OS and `C:\student` on Microsoft Windows.

Complete the following steps:

Tool Reference – Server Resources: Databases: Starting JavaDB

1. If required, start the JavaDB database server.

   Use the Tools menu of the IDE to start the database server:

   `Tools > JavaDB Database > Start`

2. Establish the JDBC™ connection to the sample database.
   a. Select the `Services` tab of the IDE.
   b. Expand the `Databases` node in the `Services` tab.
   c. Right click on the following JDBC connection and select the Connect menu option.

      ```
      jdbc:derby://localhost:1527/sample [app on APP]
      ```

      If prompted, the password is `app`.
3. Open the pre-created `AuctionApp` project:
   a. Select the `Projects` tab of the IDE.
   b. Click the File menu and select the `Open Projects` menu option.
   c. In the ensuing Open Project dialog, browse to the `exercises/mod02_auction` directory, select the `AuctionApp` project.
   d. Ensure the following check boxes are selected:
      - `Open as Main Project`
      - `Open Required Projects`
   e. Click the `Open Project` button.
4. Build the `AuctionApp` application:
   a. Select the Projects tab of the IDE.
   b. Right click the `AuctionApp` project and select the `Build Project` menu option.
5. Deploy the AuctionApp application:
   a. Select the Projects tab of the IDE.
   b. Right click the `AuctionApp` project and select the `Undeploy and Deploy` menu option.

Task 2 – Using the Auction System

In this task, you familiarize yourself with the `add-auction` and `place-bid` functionality of the auction application.

Complete the following steps:

1. Use the `Services` tab of the IDE to verify that the application server has created `AUCTION`, `AUCTIONUSER`, `BID` and `ITEM` tables.

   Verify that each table is empty.

   In the `Services` tab of the IDE, expand the following connection to reveal the `Tables` node under the connection:

   `java:derby://localhost:1527/sample [app on App]`

   Expand the `Tables` node under the connection.

   Right click on the `ITEM` table icon under the `Tables` node and select the `View Data` menu option. This causes the following SQL command to execute.

   ```
   select * from APP.ITEM
   ```

   You should observe an output tab showing the `ITEM` table with column headers and no rows.

   Repeat for `AUCTION`, `AUCTIONUSER` and `BID` tables.

2. Use a browser to load the following URL:

   `http://localhost:8080/AuctionApp/AuctionApp-app-client`

   The execution of this step might cause the browser to display the following dialog boxes:

   a. Accept the use of the `javaws` dialog box.

      This dialog box requests whether the user wants to open the application with `javaws`. Answer yes to this question.

   b. A security dialog box.

      This dialog box warns you that the server does not have a valid digital signature. To continue, ignore this warning and select the "`Always trust content from this publisher`" check box.

3. You should observe Java Webstart loading the `AuctionApp TestClient` GUI.

   You should observe the following message output on the `AuctionApp Log` window of the GUI:

```
Received hello on startup
```

4. Use the Services tab of the IDE to verify that the application server has populated the AUCTIONUSER table.

5. Enter the following string in the AuctionApp TestClient GUI window.

```
addAuction 5000 100 5 car car.jpg 2
```

You should see the following output:

```
User input: addAuction 5000 100 5 car car.jpg 2
User #2 added an auction with the ID = 6(Note: Actual ID might vary).
```

6. Use the Services tab of the IDE to verify that the application server has written to the AUCTION and ITEM tables. They should contain an entry for the auction and item you created.

7. Enter the following string in the AuctionApp TestClient GUI window.

```
placeBid 5 1 5500
```

You should see the following output:

```
User input: placeBid 5 3 5500
User #3 has placed a bid
```

8. Use the Services tab of the IDE to verify that the application server has written to the BID table. It should contain an entry for the bid you created.

Task 3 – Undeploying the Auction System

In this task, you undeploy the auction application from the application server.

Complete the following steps:

1. Perform the following steps in the `Services` tab of the IDE.
   a. Expand the following nodes to reveal the `AuctionApp` application.

```
Servers > GlassFish V2 > Applications > Enterprise Applications
```

   b. Right click `AuctionApp` and select the `Undeploy` menu option.
2. Close the `AuctionApp` project in the Projects tab of the IDE

   Right click each of the following projects and select the Close Project menu option:

   - `AuctionApp`
   - `AuctionApp-app-client`
   - `AuctionApp-ejb`

Exercise 3: Completing Review Questions

In this exercise, you answer questions about the material covered in Module 2 of the Student Guide.

Complete the following questions:

1. List the domain objects of the auction application?
2. Name the session bean that serves as the session facade to the auction client?

Exercise Summary

Discussion – Take a few minutes to discuss what experiences, issues, or discoveries you had during the lab exercise.

- Experiences
- Interpretations
- Conclusions
- Applications

Lab 3

Implementing EJB 3.0 Session Beans

Objectives

Upon completion of this lab, you should be able to:

- Create the `AuctionApp` project
- Create the `AuctionManager` session bean and corresponding `TestClient` class
- Complete the review questions

Exercise 1 – Creating the AuctionApp Project

In this exercise, you create a session façade session for the auction application.

This exercise contains the following sections:

- "Task 1 – Creating the Project"
- "Task 2 – Copying Files"

Preparation

This exercise assumes the application server is installed and running.

Tool Reference – Tool references used in this exercise:

- Java EE Development: Enterprise Application Projects: Creating Enterprise Application Projects
- Java Development: Java Application Projects: Opening Projects
- Java EE Development: Enterprise Application Projects: Building Java EE Applications
- Java EE Development: Enterprise Application Projects: Verifying Java EE Applications
- Java EE Development: Enterprise Application Projects: Deploying Java EE Applications
- Java Development: Java Application Projects: Running Projects
- Server Resources: J2EE Application Servers: Examining Server Log Files
- Java Development: Modifying Project Libraries
- Java Development: Java Classes: Modifying Java Classes: Adding Fields
- Java Development: Java Classes: Opening Java Classes
- Java Development: Other Files: Opening Files
- Java EE Application Servers: Starting Application Servers
- Server Resources: Databases: Starting JavaDB

Task 1 – Creating the Project

In this task, you configure the application server and JavaDB database server, and create necessary projects for the Auction system application.

Note – Your home directory `$HOME` should be `/export/home/student` on Solaris OS and `C:\student` on Microsoft Windows.

Tool Reference – Java EE Application Servers: Starting Application Servers

Complete the following steps:

1. Start the application server if you have shut it down.

   Use the `Services` tab of the IDE. Expand the Servers node. Right click `GlassFish V2` and select the `Start` menu option.

Tool Reference – Server Resources: Databases: Starting JavaDB

2. Start the JavaDB database server.

   Use the Tools menu of the IDE to start the database server:

   `Tools > JavaDB Database > Start`

Tool Reference – Java EE Development: Enterprise Application Projects: Creating Enterprise Application Projects

3. Create the `AuctionApp` Enterprise Application project.

   Use the File menu of the `Projects` tab of IDE to obtain a New Enterprise Application dialog.

`File > New Project > Enterprise > Enterprise Application`

   Create an enterprise project with the following characteristics:

   Project Name: **`AuctionApp`**

   Project Location: ***`$HOME`*`/project`**

   Project Folder: ***`$HOME`*`/project/AuctionApp`**

   Server: **`Sun Java System Application Server`**

   J2EE Version: **`Java EE 5`**

   Create EJB Module: **`Check`** **`AuctionApp-ejb`**

   Create Web Application Module: **`Uncheck`**

Create Application Client Module: **`Check AuctionApp-app-client`**

Main Class: **auctionapp.TestClient**

Task 2 – Copying Files

In this task, you create subdirectories and copy files needed for the AuctionApp project.

Complete the following steps:

1. Using the Files tab of the IDE, traverse to the `java` subdirectory with the path:

   `AuctionApp-ejb/src/java`

   Figure 3-1 shows the source directory structure and the location of the `java` directory.

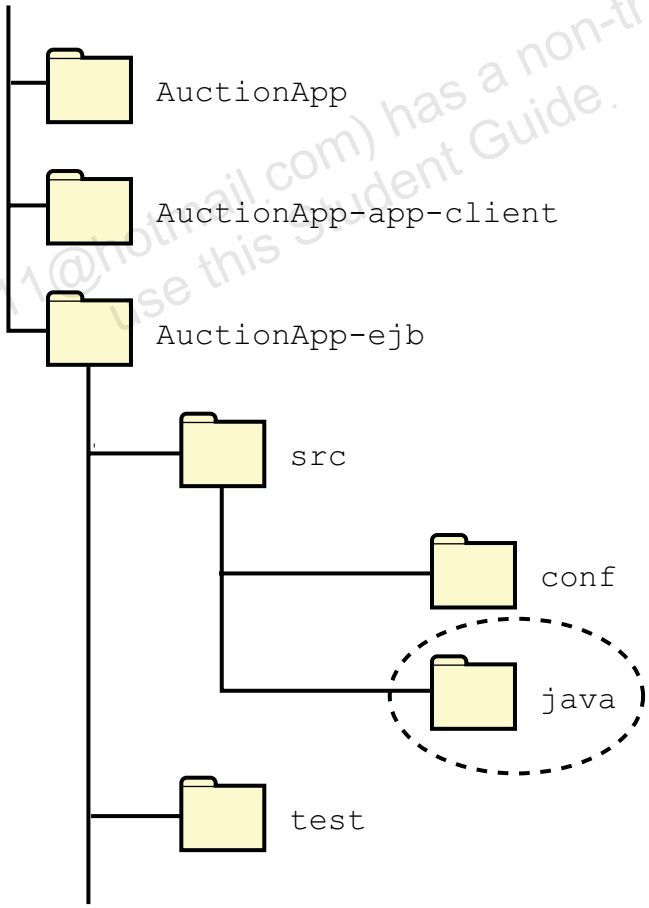

Figure 3-1 Source Directory Structure Before Copying the `auctionsystem` Directory

2. Open the Favorites tab of the IDE to verify that it contains a folder named `exercises`. If it is *missing,* add the `$HOME/exercises` directory to the Favorites tab of the IDE.
   a. Obtain an `Add to Favorites` dialog box by right clicking on the Favorites tab.
   b. Select the `exercises` folder in the folder browser section of the dialog box.
   c. Click the `Add` button.
3. Using the Favorites tab, copy the `auctionsystem` folder (this automatically includes all the subfolders of `auctionsystem`). The path to the `auctionsystem` directory is:

   ```
   exercises/mod03_session/exercise1/auctionsystem
   ```

4. Paste the `auctionsystem` folder into the `java` folder in the Files tab of the IDE. The path to the `java` directory is:

   ```
   AuctionApp-ejb/src/java
   ```

5. Expand the `java` directory and verify that you have the subdirectories shown in Figure 3-2.

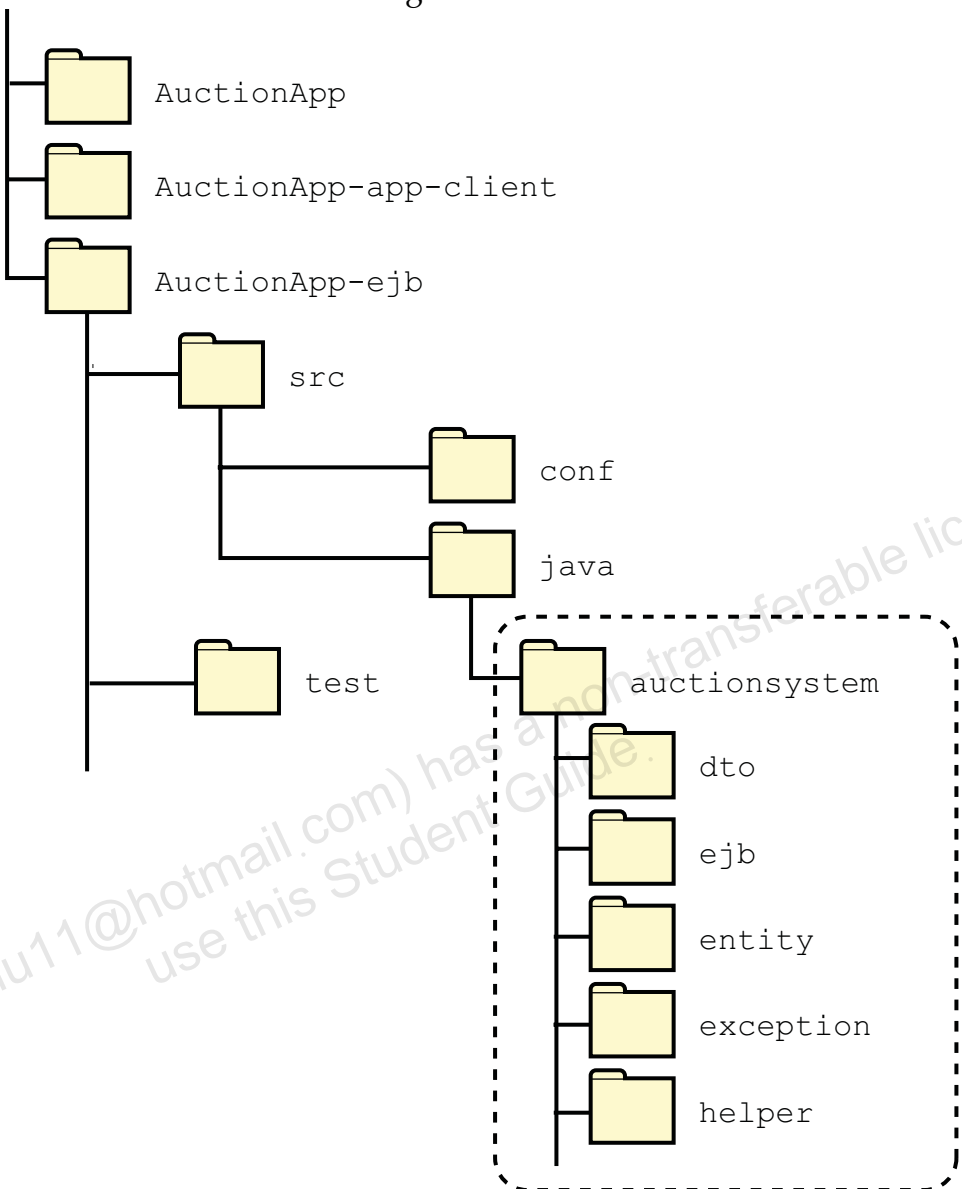

Figure 3-2 Source Directory Structure After Copying the `auctionsystem` Directory

6. Using the Files tab of the IDE, open the `exception` subdirectory. Verify that you have the following classes in the `exception` directory.

   ```
   AuctionException.java
   BidTooLowException.java
   CloseException.java
   CreditCardException.java
   OutbidException.java
   PlaceBidException.java
   ```

7. Using the Files tab of the IDE, open the `dto` subdirectory. Verify that you have the following classes in the `dto` directory.

   ```
   BidStatusMessage.java
   PlaceBidMessage.java
   ```

8. Build the `Auction-ejb` project. Correct any errors you encounter.

   From the `Projects` tab of the IDE, right click the `Auction-ejb` project > select `Build Project` menu option.

Exercise 2 – Creating a Session Bean

In this exercise, you create a session bean that functions as the session facade for the auction application.

This exercise contains the following sections:

- "Task 1 – Creating the `AuctionManager` Session Bean"
- "Task 2 – Implementing the `TestClient` Class"
- "Task 3 – Building and Deploying the `AuctionApp` Application"
- "Task 4 – Testing the `AuctionApp` Application"

Preparation

This exercise assumes the application server is installed and running.

Tool Reference – Tool references used in this exercise:

- Java Development: Java Application Projects: Opening Projects
- Java EE Development: Enterprise Application Projects: Building Java EE Applications
- Java EE Development: Enterprise Application Projects: Verifying Java EE Applications
- Java EE Development: Enterprise Application Projects: Deploying Java EE Applications
- Server Resources: J2EE Application Servers: Examining Server Log Files
- Java Development: Java Application Projects: Modifying Project Libraries
- Java Development: Java Classes: Modifying Java Classes: Adding Fields
- Java Development: Java Classes: Modifying Java Classes: Adding Methods
- Java Development: Java Classes: Opening Java Classes
- Java Development: Other Files: Opening Files
- Java EE Application Servers: Starting Application Servers
- Server Resources: Databases: Starting JavaDB

Task 1 – Creating the AuctionManager Session Bean

In this task, you create and implement the `AuctionManager` session bean. Figure 3-3 shows the components of the `AuctionManager` session bean created in this task. In this module, you implement the `communicationTest` method. For the other methods, you provide the minimal functionality required to compile the classes. In subsequent modules, you enhance the functionality of the other methods.

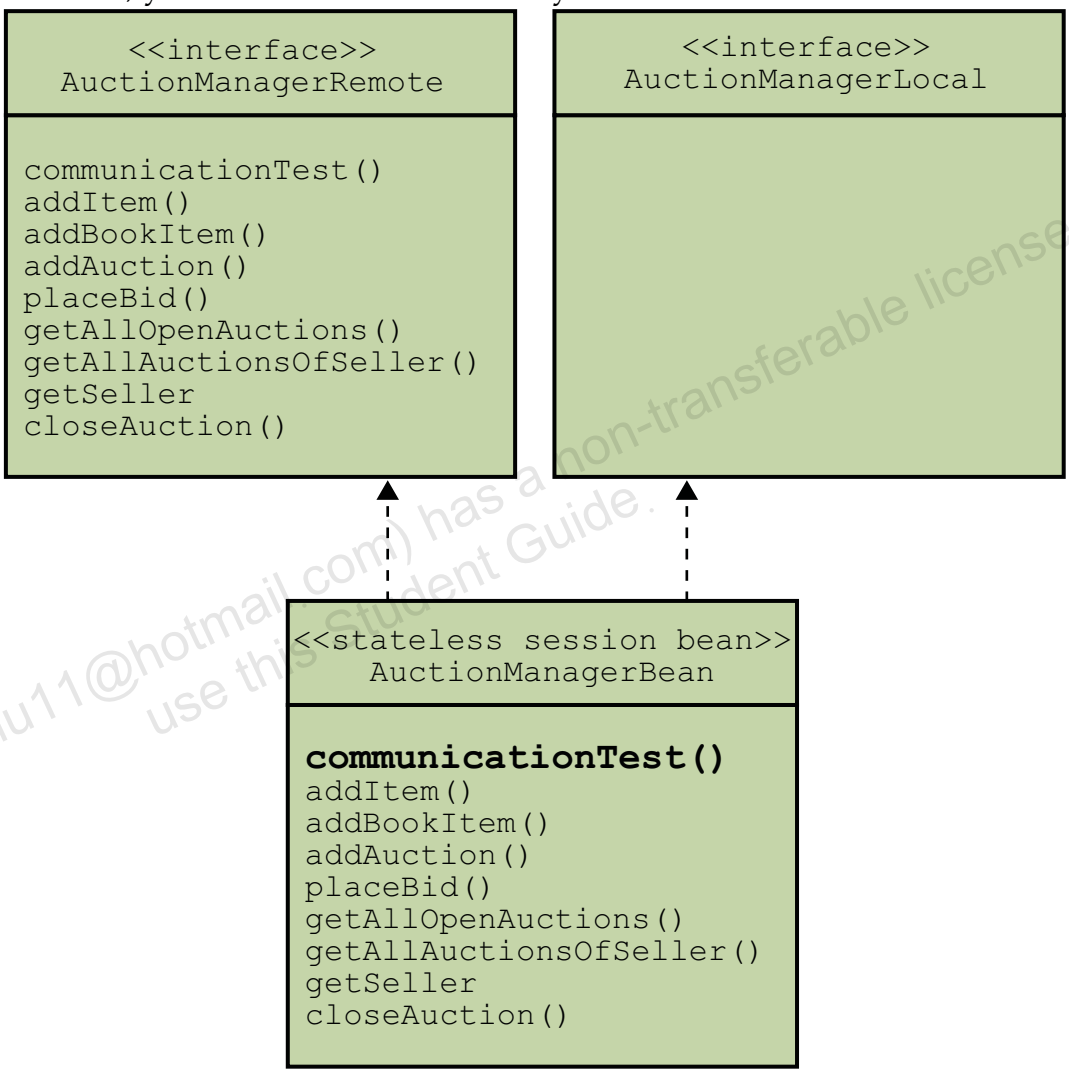

Figure 3-3 Classes Created in Task 1

Complete the following steps:

1. Create the `AuctionManager` stateless session bean in the `AuctionApp-ejb` project.

   a. Use the `Projects` tab of the IDE to open a New Session Bean dialog box:

      Right click `AuctionApp-ejb` > `New` > `Session Bean`

   b. Create the session bean with the following characteristics:

   EJB Name: **`AuctionManager`**

   Project: **`AuctionApp-ejb`**

   Location: **`Source Packages`**

   Package: **`auctionsystem.ejb`**

   Session Type: **`Stateless`**

   Create Interface

      Remote: **`Yes`**

      Local: **`Yes`**

2. Open the `AuctionManagerRemote.java` file in the source code editor window.

3. Import the following classes:

```
java.util.Date;
java.util.Collection;
auctionsystem.exception.AuctionException;
```

4. Add the following methods to the `AuctionManagerRemote` interface.

```
public String communicationTest(String message);
public Object addItem(String itemDesc, String itemImage);
public Object addBookItem(String itemDesc, String itemImage,
  String title, String author);
public int addAuction(double startAmount, double increment,
  Date closeTime, String itemDesc, String itemImage, Integer sellerID);
public void placeBid(Integer auctionID, Integer bidderID,
  double bidAmount) throws AuctionException;
public Collection getAllOpenAuctions();
public Collection getAllAuctionsOfSeller(Integer sellerID);
public Object getSeller(String displayName);
public void closeAuction(Integer auctionID) throws AuctionException;
```

5. Open the `AuctionManagerBean.java` file in the source code editor window.

6. Use the IDE to create empty method implementations of all the `AuctionManagerRemote` interface methods.
   a. Click the line of source code that contains the keyword `implements`.
   b. A light bulb should appear to the left of the line. Press the `Alt` and `Enter` keys.
   c. Select the "`Implement all abstract methods`" pop-up menu item. This causes the IDE to produce empty implementations for all the `AuctionManagerRemote` interface methods.
7. Code the body of the `communicationTest` method. You can use the following code as your implementation.

```
public String communicationTest(String message) {
  System.out.println("AuctionManager.communicationTest: " + message);
  return "Received " + message;
}
```

8. Build the `AuctionApp` project. Correct any errors you encounter.

   IDE `Projects` tab > right click `AuctionApp` > `Build Project`

9. Verify the `AuctionApp` project.

   IDE `Projects` tab > right click `AuctionApp` > `Verify Project`

Note – Whenever you verify a project, you should select the `Failures and Warnings` display option in the output tab. You should fix any failures and warnings displayed before proceeding.

Task 2 – Implementing the `TestClient` Class

In this task, you create and implement the `TestClient` class. The version you create in this module has the minimal functionality required to test the invocation of the `communicationTest` method of the `AuctionManager` session bean instance executing in a Java EE application server.

Figure 3-4 illustrates the `TestClient` class you create in this task.

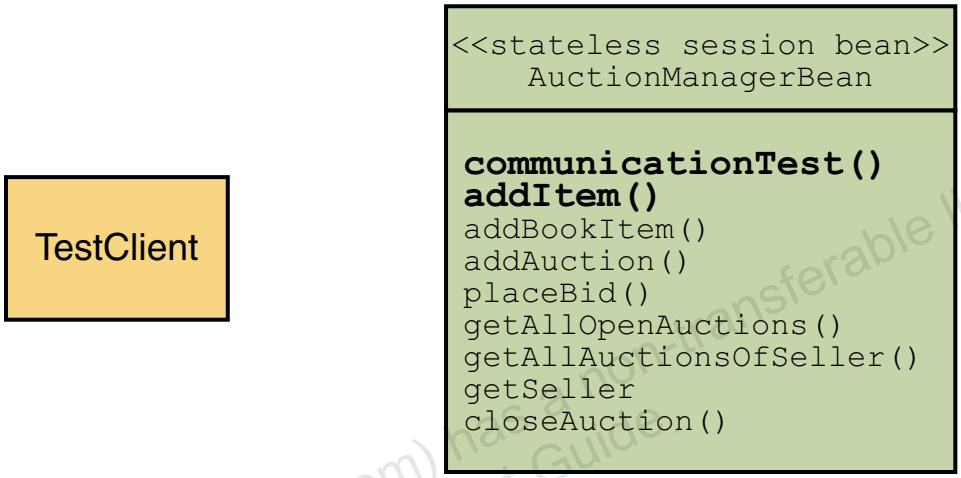

Figure 3-4 The `TestClient` Class

Complete the following steps:

1. Open the `TestClient.java` class in the source code editor window.

   Use the Files tab of the IDE and locate and open the `TestClient.java` file.

```
AuctionApp-app-client > src > java > auctionapp > TestClient.java
```

2. Add import statements in the `TestClient` class to import all classes from the following packages.

```
auctionsystem.ejb
javax.ejb
```

3. Add an injected private static instance variable of type `AuctionManagerRemote` with the name `auctionManager`.

```
@EJB private static AuctionManagerRemote auctionManager;
```

4. Add code in the main method to invoke the `communicationTest(String message)` method of the `AuctionManager` session bean. For example:

```
String message = "hello";
System.out.println("Sending " + message);
String reply = auctionManager.communicationTest(message);
System.out.println(reply);
```

5. Build the `AuctionApp` project. Correct any errors you encounter.
6. Verify the `AuctionApp` project

Note – Building the `AuctionApp` project automatically builds (as required) the projects it depends on, namely the `AuctionApp-ejb` project and the `AuctionApp-app-client` project.

Task 3 – Building and Deploying the AuctionApp Application

In this task, you build, verify, and finally deploy the AuctionApp EAR file on a Java EE application server.

Complete the following steps:

1. Using the Files tab of the IDE, check that the following file has been created:

`AuctionApp-app-client/dist/AuctionApp-app-client.jar`

   The `AuctionApp-app-client.jar` file is the client JAR module of the `AuctionApp` application.

2. Using the Files tab of the IDE, check that the following file has been created:

`AuctionApp-app-ejb/dist/AuctionApp-ejb.jar`

   The `AuctionApp-app-ejb.jar` file is the EJB-JAR module of the `AuctionApp` application. It contains the server-side components.

3. Using the Files tab of the IDE, verify that the following file has been created:

`AuctionApp/dist/AuctionApp.ear`

   The `AuctionApp-app.ear` file is the deployable application EAR module of the `AuctionApp` application. It includes both the client and the server-side components.

4. Deploy the `AuctionApp` application.

   IDE `Projects` tab > right click `AuctionApp` > Undeploy and `Deploy Project`

Note – During deployment, you might observe a `java.io.FileException` associated with deployment of the `AuctionApp-ejb.jar` file. You can ignore the error, if the application server ignores it and continues with the deployment.

Task 4 – Testing the AuctionApp Application

In this task, you test the AuctionApp application.

Complete the following steps:

1. Log in to the application server's admin console.
   a. Enter the following URL into a web browser:

      localhost:4848

   b. Enter the following user name and password into the ensuing login page:

      User: **admin**

      Password: **adminadmin**

2. Using the application server's admin console, download the client stub file AuctionApp-app-client.jar to $HOME/project/client directory.
   a. Use the application server's admin console and expand the following nodes to view the Enterprise Application page for the AuctionApp application:

```
Applications > Enterprise Applications > AuctionApp
```

   b. Select the Download Client Stubs action to download the AuctionApp-app-client.jar file to the following directory:

      $HOME/project/client

3. Obtain a terminal window and set the current directory of the terminal window to the same directory where you downloaded the client stub file:

   $HOME/project/client

4. Execute the following command on the terminal window:

```
/opt/glassfish-v2/bin/appclient -client AuctionApp-app-client.jar.jar
```

Note – The appclient executable is located in the bin directory of the application server. The default location of the bin directory on a Solaris OE installation is:

/opt/glassfish-v2/bin

The location of the application server on your system might be different.

You should observe the following message output on your terminal window:

```
Sending hello
Received hello
```

5. Examine the output on the IDE output window marked `GlassFish V2`. Scroll the server log to the end of the log and you should see the following line:

   ```
   AuctionManager.communicationTest: Hello
   ```

Exercise 3: Completing Review Questions

In this exercise, you answer questions about the material covered in Module 3 of the Student Guide.

Complete the following questions:

1. Name the two types of session beans?
2. To declare a session EJB as a remote EJB, list two possible locations where you can place the @Remote annotation?
3. List two event related annotations that have meaning in a stateful session bean but not in a stateless session bean?

Exercise Summary

Discussion – Take a few minutes to discuss what experiences, issues, or discoveries you had during the lab exercise.

- Experiences
- Interpretations
- Conclusions
- Applications

Lab 4

Implementing Entity Classes: The Basics

Objectives

Upon completion of this lab, you should be able to:

- Create the `AuctionApp` entity classes
- Implement the `additem` use case
- Populate the `AuctionUser` table
- Implement the `addAuction` use case
- Complete the review questions

Exercise 1: Creating the AuctionApp Entity Classes

This exercise contains the following sections:

- Task 1 – Creating the `AuctionDB` Database
- Task 2 – Creating a Persistence Unit
- Task 3 – Creating the `Item` Entity
- Task 4 – Creating the `AuctionUser` Entity
- Task 5 – Creating the `Auction` Entity
- Task 6 – Creating the `Bid` Entity

Tool Reference – Tool references used in this exercise:

- Java EE Development: Enterprise Application Projects: Creating Enterprise Application Projects
- Java Development: Java Application Projects: Opening Projects
- Java EE Development: Enterprise Application Projects: Building Java EE Applications
- Java EE Development: Enterprise Application Projects: Verifying Java EE Applications
- Java EE Development: Enterprise Application Projects: Deploying Java EE Applications
- Java Development: Java Application Projects: Running Projects
- Server Resources: J2EE Application Servers: Examining Server Log Files
- Java Development: Modifying Project Libraries
- Java Development: Java Classes: Modifying Java Classes: Adding Fields
- Java Development: Java Classes: Opening Java Classes
- Java Development: Other Files: Opening Files
- Java EE Application Servers: Starting Application Servers
- Server Resources: Databases: Starting JavaDB
- Server Resources: Databases: Creating a Java DB Database
- Server Resources: Databases: Connecting to Databases
- Java EE Development: Persistence Modules: Creating Persistence Units

Task 1 – Creating the AuctionDB Database

Complete the following steps:

1. Start the Java DB database server if it is stopped.

   To perform this step, use the Tools menu as follows:

   `Tools > Java DB Database > Start Java DB Server`

2. Open the Create a new Java DB database dialog box.

   To perform this step, use the Tools menu as follows:

   `Tools menu > Java DB Database > Create Java DB Database`

   Enter the following properties:

   - Database Name: **`AuctionDB`**
   - User Name: **suned**
   - Password: **suned**

3. Connect to the newly created database using the `java:derby://localhost:1527/AuctionDB` connection.

   Expand the Databases node in `Services` tab of the IDE.

   Right click the following database connection and select the Connect menu option.

`java:derby://localhost:1527/AuctionDB [suned on SUNED]`

Task 2 – Creating a Persistence Unit

In this task, you create a new persistence unit. Figure 4-1 illustrates a new empty persistence unit you create in this task. In subsequent tasks, you define new persistence entity classes for inclusion in the persistence unit.

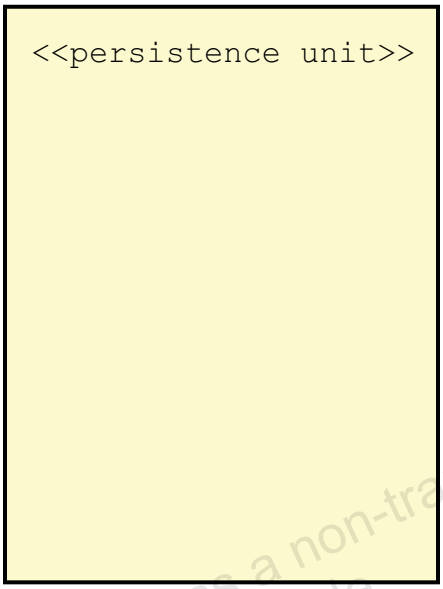

Figure 4-1 An Empty Persistence Unit

Complete the following steps:

1. Obtain the New Persistence Unit dialog box.
   a. Right-click the `AuctionApp-ejb` node in the Projects tab and choose `New` > `Other` to open the New File menu.
   b. From the `Persistence` category, select `Persistence Unit` and click Next.
2. Enter the following information into the New Persistence Unit dialog box:

   Persistence Unit Name: **`AuctionApp-ejbPU`**

   Persistence Provider: **`TopLink(default)`**

   Data Source: **`New Data Source...`**

   - JNDI Name: **`jdbc/auctionDB`**
   - Database Connection: **`java:derby://localhost:1527/AuctionDB [suned on SUNED]`**

   Use Java Transaction APIs: **`Yes`**

   Table Generation Strategy: **`Drop and Create`**

3. The completion of the previous step opens the `persistence.xml` file in the source editor window. By default, the `persistence.xml` file opens using a persistence unit editor dialog wizard.

   Click the XML tab to see an XML view of the `persistence.xml` file.

   Examine the `peristence.xml` file. The listing contained in Code 4-1 is similar to what you have created.

Code 4-1 Listing of the `persistence.xml` File

```
1   <?xml version="1.0" encoding="UTF-8"?>
2   <persistence version="1.0"
       xmlns="http://java.sun.com/xml/ns/persistence"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
       http://java.sun.com/xml/ns/persistence/persistence_1_0.xsd">
3     <persistence-unit name="AuctionApp-ejbPU" transaction-type="JTA">
4       <provider>
5         oracle.toplink.essentials.PersistenceProvider
6       </provider>
7       <jta-data-source>jdbc/auctionDB</jta-data-source>
8       <properties>
9         <property
10         name="toplink.ddl-generation" value="drop-and-create-tables"/>
11      </properties>
12    </persistence-unit>
13  </persistence>
```

Task 3 – Creating the Item Entity

In this task, you create the Item entity class. Figure 4-2 illustrates the persistence unit populated with the Item entity you create in this task.

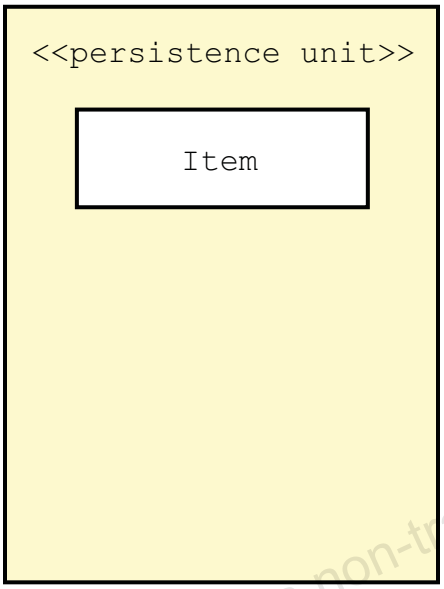

Figure 4-2 Persistence Unit Populated With the Item Entity

Complete the following steps:

1. Open the New Entity Class dialog box.

   Right-click the AuctionApp-ejb project node and choose New > Entity class.

2. Enter the following information in the New Entity Class dialog box:

   Class Name: **Item**

   Project: **AuctionApp-ejb**

   Location: **Source Packages**

   Package: **auctionsystem.entity**

   Primary key Type: **Integer**

3. Add the following fields to the Item class.

```
private String description;
private String image;
```

4. Use the source editor to encapsulate the fields (you added in the previous step) with public getter and setter methods.

   Right-click in the Source Editor and choose `Refactor` > `Encapsulate` fields to generate getters and setters for each of the fields. In the Encapsulate Fields dialog box, make sure that the getter and setter check boxes are selected for all of the fields.

5. Add the following constructors to the `Item` class.

```
public Item () {
}

public Item (String description, String image) {
  setDescription(description);
  setImage(image);
}
```

6. Build the `AuctionApp` project. Correct any errors.
7. Open the `AuctionManagerBean` class in a source editor window.
8. Import the following classes.

```
import javax.persistence.EntityManager;
import javax.persistence.PersistenceContext;
import auctionsystem.entity.Item;
```

9. Add the following instance variable.

```
@PersistenceContext private EntityManager em;
```

10. Verify the `AuctionApp` project.
11. Deploy the `AuctionApp` application.
12. Verify that the application server has created an `Item` table.

    In the `Services` tab of the IDE, right click the `Tables` node under the following connection:

    `java:derby://localhost:1527/AuctionDB [suned on SUNED]`

    Select the `refresh` menu option. You should now see a node marked `ITEM`.

    Right click on the `ITEM` table icon and select the view data menu option. This causes the following SQL command to execute.

    ```
    select * from SUNED.ITEM
    ```

    You should observe an output tab showing the `ITEM` table with column headers and no rows.

Task 4 – Creating the AuctionUser Entity

In this task, you create the `AuctionUser` entity class. Figure 4-3 illustrates the persistence unit populated with the `AuctionUser` entity you create in this task.

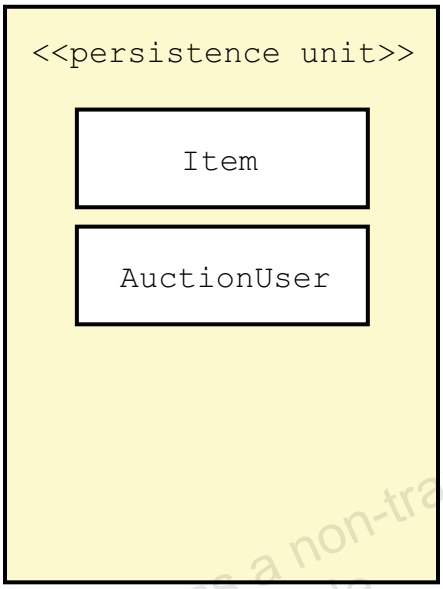

Figure 4-3 Persistence Unit Populated With the `AuctionUser` Entity

The steps in this task are a repeat of the previous task. If time is a factor, you can *bypass* this task by performing the following steps.

1. Use the IDE to copy the file `AuctionUser.java` from `exercises/mod04_entity1/exercise1` to the `AuctionApp-ejb/src/java/auctionsystem/entity directory.`
2. Proceed to Step 6 on page 4-9.

Complete the following steps:

1. Open the New Entity Class dialog box.

   Right-click the `AuctionApp-ejb` project node and choose `New` > `Entity class`.

2. Enter the following information in the New Entity Class dialog box:

   Class Name: **AuctionUser**

   Project: **AuctionApp-ejb**

   Location: **Source Packages**

   Package: **auctionsystem.entity**

Primary key Type: **Integer**

3. Add the following fields to the AuctionUser class.

```
private String displayName;
private String email;
```

4. Use the source editor to encapsulate the fields (you added in the previous step) with public getter and setter methods.

   Right-click in the Source Editor and choose Refactor > Encapsulate fields to generate getters and setters for each of the fields. In the Encapsulate Fields dialog box, make sure that the getter and setter check boxes are selected for all of the fields

5. Add the following constructors to the AuctionUser class

```
public AuctionUser () {
}

public AuctionUser (String displayName, String email) {
  setDisplayName(displayName);
  setEmail(email);
}
```

6. Build the AuctionApp project. Correct any errors you encounter.
7. Verify the AuctionApp project.
8. Deploy the AuctionApp application.
9. Verify that the application server has created an AUCTIONUSER table.

   In the Services tab of the IDE, right click the Tables node under the following connection:

   java:derby://localhost:1527/AuctionDB [suned on SUNED]

   Select the refresh menu option. You should now see a node marked AUCTIONUSER.

   Right click on the AUCTIONUSER table icon and select the view data menu option. This causes the following SQL command to execute.

   ```
   select * from SUNED.AUCTIONUSER
   ```

   You should observe an output tab showing the AUCTIONUSER table with column headers and no rows.

Task 5 – Creating the Auction Entity

In this task, you create the `Auction` entity class. Figure 4-4 illustrates the persistence unit populated with the `Auction` entity you create in this task.

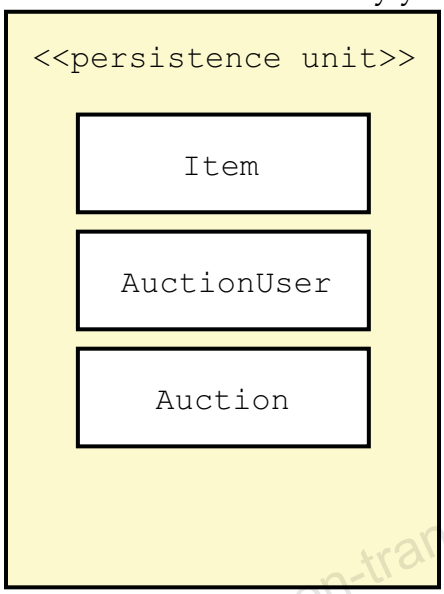

Figure 4-4 Persistence Unit Populated With the `Auction` Entity

The steps in this task are a repeat of the previous task. If time is a factor, you can *bypass* this task by performing the following steps.

1. Use the IDE to copy the file `Auction.java` from the `exercises/mod04_entity1/exercise1` to the `AuctionApp-ejb/src/java/auctionsystem/entity` directory.
2. Proceed to Step 7 on page 4-12.

Complete the following steps:

1. Open the New Entity Class dialog box.

   Right-click the `AuctionApp-ejb` project node and choose `New` > `Entity class`.

2. Enter the following information in the New Entity Class dialog:

   Class Name: **Auction**

   Project: **AuctionApp-ejb**

   Location: **Source Packages**

   Package: **auctionsystem.entity**

   Primary key Type: **Integer**

3. Use the source editor to import the following classes to the `Auction` class.

```
import javax.persistence.Temporal;
import javax.persistence.TemporalType;
import javax.ejb.EJBException;
import java.util.Date;
import java.util.Calendar;
```

4. Add the following fields to the `Auction` class:

```
private double startAmount;
private double increment;
private String status;
private Date openTime;
private Date closeTime;
```

Note – Date is a temporal type and must be annotated with the `Temporal(TemporalType.DATE) annotation.`

5. Use the source editor to encapsulate the fields (you added in the previous step) with public getter and setter methods.

   Right-click in the Source Editor and choose `Refactor` > `Encapsulate` fields to generate getters and setters for each of the fields. In the Encapsulate Fields dialog box, make sure that the getter and setter check boxes are selected for all of the fields.

   Annotate the `getOpenTime` and `getCloseTime` methods with the following annotation:

```
@Temporal(TemporalType.DATE)
```

Note – Date is a temporal type and must be annotated with the `Temporal(TemporalType.DATE) annotation.`

6. Add the following constructors to the `Auction` class.

```
public Auction() {
}

public Auction(double startAmount, double increment, Date closeTime,
  Item item, AuctionUser seller){
  setOpenTime(Calendar.getInstance().getTime());
  if (openTime.after(closeTime)) {
    throw new EJBException("To open this auction successfully, " +
      "a later close time must be chosen");
  }
  setCloseTime(closeTime);
```

```
  setStartAmount(startAmount);
  setIncrement(increment);
  setStatus("OPEN");
}
```

7. Build the AuctionApp project. Correct any errors you encounter.
8. Verify the AuctionApp project.
9. Deploy the AuctionApp application.
10. Verify that the application server has created an AUCTION table.

    Right click the Tables node in the Services tab of the IDE and select the refresh menu option. You should now see a node marked AUCTION.

    Right click on the AUCTION table icon and select the view data menu option. This causes the following SQL command to execute.

    ```
    select * from SUNED.AUCTION
    ```

    You should observe an output tab showing the AUCTION table with column headers and no rows.

Task 6 – Creating the Bid Entity

In this task, you create the Bid entity class. Figure 4-5 illustrates the persistence unit populated with the Bid entity you create in this task.

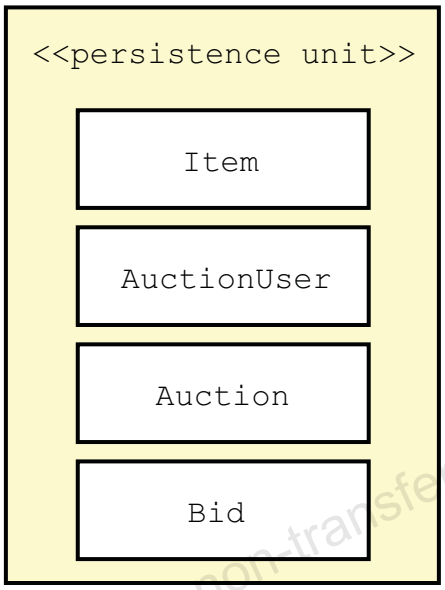

Figure 4-5 Persistence Unit Populated With the Bid Entity

The steps in this task are a repeat of the previous task. If time is a factor, you can *bypass* this task by performing the following steps.

1. Use the IDE to copy the file Bid.java from exercises/mod04_entity1/exercise1 to the AuctionApp-ejb/src/java/auctionsystem/entity directory.
2. Proceed to Step 7 on page 4-14.

Complete the following steps:

1. Open the New Entity Class dialog box.

   Right-click the `AuctionApp-ejb` project node and choose `New` > `Entity class`.

2. Enter the following information in the New Entity Class dialog:

   Class Name: **`Bid`**

   Project: **`AuctionApp-ejb`**

   Location: **`Source Packages`**

   Package: **`auctionsystem.entity`**

   Primary key Type: **`Integer`**

3. Use the source editor to import the following classes to the `Bid` class.

```
import javax.persistence.Temporal;
import javax.persistence.TemporalType;
import java.util.Date;
import java.util.Calendar;
```

4. Use the source editor to add the following fields to the `Bid` class.

```
private double amount;
private Date bidTime;
private String approval;
```

5. Use the source editor to encapsulate the fields (you added in the previous step) with public getter and setter methods.

   Right-click in the Source Editor and choose `Refactor` > `Encapsulate` fields to generate getters and setters for each of the fields. In the Encapsulate Fields dialog box, make sure that the getter and setter check boxes are selected for all of the fields

   Annotate the `getBidTime` method with the following annotation:

   `@Temporal(TemporalType.DATE)`

6. Add the following constructors to the `Bid` class.

```
public Bid(){
}

public Bid(double amount, String authorization, Auction auction,
  AuctionUser bidder){
  setAmount(amount);
  setApproval(authorization);
  setBidTime(Calendar.getInstance().getTime());
}
```

7. Build the `AuctionApp` project. Correct any errors you encounter.

8. Verify the AuctionApp project.
9. Deploy the AuctionApp application.
10. Verify that the application server has created a BID table.

    Right click the Tables node in the Services tab of the IDE and select the refresh menu option. You should now see a node marked BID.

    Right click the BID table icon and select the view data menu option. This causes the following SQL command to execute.

    ```
    select * from SUNED.BID
    ```

    You should observe an output tab showing the BID table with column headers and no rows.

Exercise 2: Implementing the addItem Use Case

This exercise contains the following sections:

- Task 1 – Updating the `AuctionManager` Session Bean
- Task 2 – Copying New Client Classes
- Task 3 – Building and Deploying the `AuctionApp` Application
- Task 4 – Testing the `AuctionApp` Application

Tool Reference – Tool references used in this exercise:

- Java EE Development: Enterprise Application Projects: Creating Enterprise Application Projects
- Java Development: Java Application Projects: Opening Projects
- Java EE Development: Enterprise Application Projects: Building Java EE Applications
- Java EE Development: Enterprise Application Projects: Verifying Java EE Applications
- Java EE Development: Enterprise Application Projects: Deploying Java EE Applications
- Java Development: Java Application Projects: Running Projects
- Server Resources: J2EE Application Servers: Examining Server Log Files
- Java Development: Modifying Project Libraries
- Java Development: Java Classes: Modifying Java Classes: Adding Fields
- Java Development: Java Classes: Opening Java Classes
- Java Development: Other Files: Opening Files
- Java EE Application Servers: Starting Application Servers
- Server Resources: Databases: Starting JavaDB
- Server Resources: Databases: Creating a Java DB Database
- Server Resources: Databases: Connecting to Databases
- Java EE Development: Persistence Modules: Creating Persistence Units

Task 1 – Updating the AuctionManager Session Bean

In this task, you update the AuctionManager session bean by implementing the addItem method. Figure 4-6 illustrates the addItem method of the AuctionManagerBean class and the Item entity that it uses.

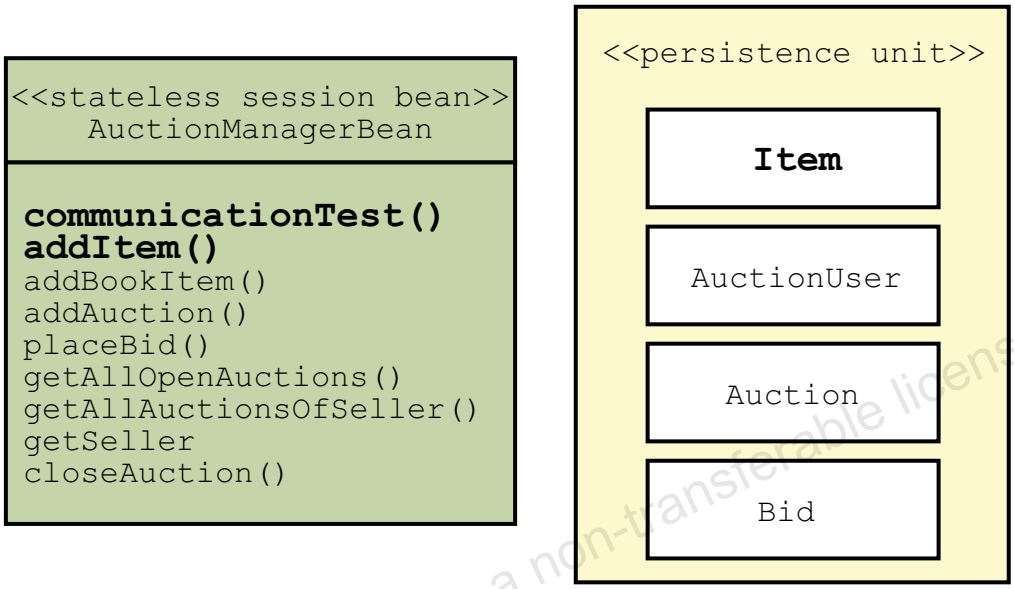

Figure 4-6 The AuctionManagerBean Class and Persistence Unit

Complete the following steps:

1. Open the AuctionManagerBean class in a source editor window.
2. Implement the following method.

```
public Object addItem(String itemDesc, String itemImage);
```

To implement the addItem method, perform the following steps.

a. To assist with diagnostics, add a print line statement that prints the following string.

```
"In AuctionManagerBean.addItem()"
```

b. Create an Item instance using the Item (String description, String image) constructor.

c. Persist the Item instance you created, using the entity managers persist method.

d. Return the Item instance.

3. Build the AuctionApp project. Correct any errors.
4. Verify the AuctionApp project.

Task 2 – Copying New Client Classes

In this task, you replace the `TestClient` class with a new version that contains a GUI.

Complete the following steps:

1. Use the `Files` tab to locate and delete the `TestClient.java` file used previously. The path to the file to delete is:

```
AuctionApp-app-client/src/java/auctionapp
```

2. Open the Favorites tab and locate the following directory:

```
exercises/mod04_entity1/exercise2
```

   Copy the following files:

   - `AuctionGui.java`
   - `GuiLogPanel.java`
   - `TestClient.java`

3. Use the Files tab to paste the files in the following directory:

```
AuctionApp-app-client/src/java/auctionapp
```

4. Build the `AuctionApp` project. Correct any errors you encounter.
5. Verify the `AuctionApp` project.

Task 3 – Building and Deploying the AuctionApp Application

In this task, you build, verify, and finally deploy the AuctionApp EAR file on a Java EE application server.

Complete the following steps:.

1. Build the AuctionApp project. Correct any errors you encounter.
2. Verify the AuctionApp project.
3. Deploy the AuctionApp application.

Task 4 – Testing the AuctionApp Application

In this task, you test the AuctionApp application. In this test, you use the TestClient class to create one or more entries in the ITEM table in the database.

Complete the following steps:

1. Use a browser to load the following URL:

   ```
   http://localhost:8080/AuctionApp/AuctionApp-app-client
   ```

2. You should observe Java Webstart loading the AuctionApp TestClient GUI.

   You should observe the following message output on the AuctionApp Log window of the GUI:

```
Received hello on startup
```

3. Enter the following string in the Auction Use Case Input text field of the AuctionApp TestClient GUI window.

```
addItem radio radio.jpg
```

   You should observe the following message output on your GUI window:

```
User input: addItem radio radio.jpg
Created auctionsystem.entity.Item[id=2]
```

4. Examine the ITEM table in the Services tab of the IDE.

   Open the Tables node in the Services tab of the IDE.

   Right click the ITEM table icon and select the view data menu option. This causes the following SQL command to execute.

   ```
   select * from SUNED.ITEM
   ```

   You should observe an output tab showing the ITEM table with column headers and a new row of data corresponding to the addItem command sent from the client.

5. Optionally, you can repeat the previous two steps to create more entries in the ITEM table.

Exercise 3: Populating the AuctionUser Table

This exercise contains the following sections:

- Task 1 –Creating the `populateAuctionUser` Method in the `AuctionManagerBean` Class
- Task 2 – Testing the `AuctionApp` Application

Tool Reference – Tool references used in this exercise:

- Java EE Development: Enterprise Application Projects: Creating Enterprise Application Projects
- Java Development: Java Application Projects: Opening Projects
- Java EE Development: Enterprise Application Projects: Building Java EE Applications
- Java EE Development: Enterprise Application Projects: Verifying Java EE Applications
- Java EE Development: Enterprise Application Projects: Deploying Java EE Applications
- Java Development: Java Application Projects: Running Projects
- Server Resources: J2EE Application Servers: Examining Server Log Files
- Java Development: Modifying Project Libraries
- Java Development: Java Classes: Modifying Java Classes: Adding Fields
- Java Development: Java Classes: Opening Java Classes
- Java Development: Other Files: Opening Files
- Java EE Application Servers: Starting Application Servers
- Server Resources: Databases: Starting JavaDB
- Server Resources: Databases: Creating a Java DB Database
- Server Resources: Databases: Connecting to Databases
- Java EE Development: Persistence Modules: Creating Persistence Units

Task 1 –Creating the populateAuctionUser Method in the AuctionManagerBean Class

In this task, you add the `populateAuctionUserTable` method to the `AuctionManagerBean` class. Figure 4-7 illustrates the `populateAuctionUserTable` method of the `AuctionManagerBean` class and the `AuctionUser` entity that it uses.

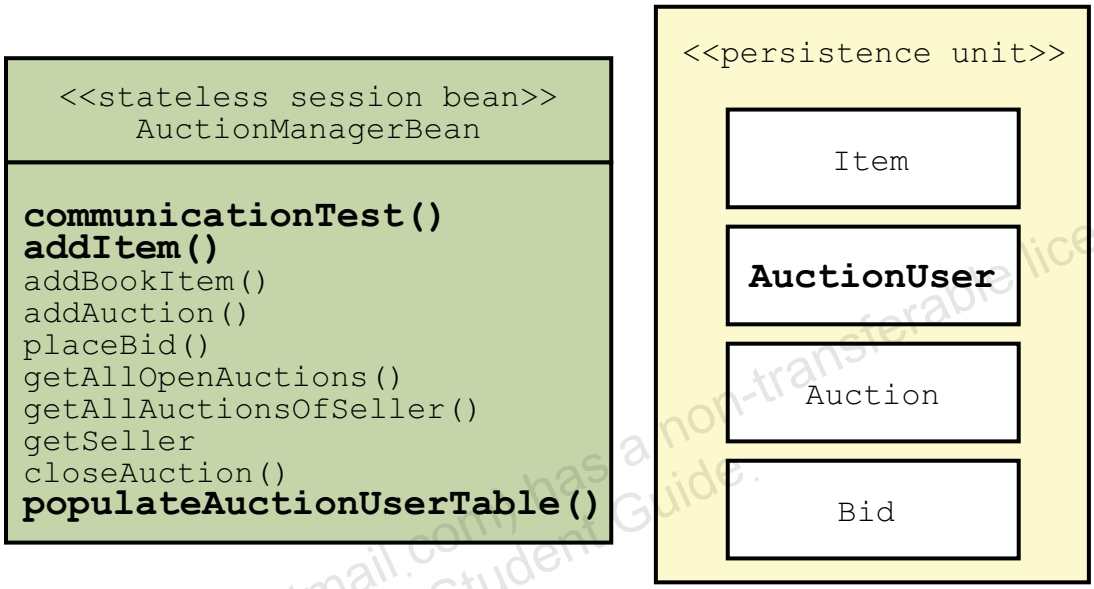

Figure 4-7 The `AuctionManagerBean` Class and Persistence Unit

Complete the following steps:

1. Open the `AuctionManagerBean` class in a source editor window.
2. Import the following classes:

```
import auctionsystem.entity.AuctionUser;
```

3. Type the `populateAuctionUserTable` method, as shown in the following code listing. You need not type the comment lines.

```
1   private void populateAuctionUserTable () {
2   // This method creates 3 rows in the AUCTIONUSER Table
3   // Note: The rows are created only if it cannot find a row with
4   //       primary key value of 2.
5     System.out.println("AuctionManagerBean.populateAuctionUserTable");
6     Object  au;
7     au = null; // see step 4 for instructions
8     if (au == null) {
9         // See step 5 for instructions
10        // Repeat step 5 instruction
```

```
11        // Repeat step 5 instruction
12    }
13  }
```

4. Replace the null in code line 7 with the `AuctionUser` entity instance obtained by invoking the entity manager's `find` method with a primary key value of 1.

   Hint: `em.find(<parameter1>, <parameter2>);`

5. Use the entity manager to create and persist a `new AuctionUser("auctionman", "auctionman@yahoo.com")`.

   Repeat the previous instruction to create and persist two more `AuctionUser` entities. Use suitable values for display name and email.

6. Add a line of code in the `communicationTest` method to invoke the `populateAuctionUserTable` method.

7. Build the `AuctionApp` project. Correct any errors.

8. Verify the `AuctionApp` project.

Note – The use of hard code in a method to populate a database table is not optimal. However, it is included because it does provide you with an additional learning opportunity to use the `EntityManager` API.
In real life, you add an additional use case "`Add Auction Users`" to populate the `AUCTIONUSER` table.

Task 2 – Testing the AuctionApp Application

Complete the following steps:

1. Deploy the AuctionApp application.
2. Verify that the application server has created an AUCTIONUSER table.

   Right click the Tables node in the Services tab of the IDE and select the refresh menu option. You should now see a node marked AUCTIONUSER.

   Right click the AUCTIONUSER table icon and select the view data menu option. This causes the following SQL command to execute.

   ```
   select * from SUNED.AUCTIONUSER
   ```

   You should observe an output tab showing the AUCTIONUSER table with column headers and no rows.
3. Use a browser to load the following URL:

   ```
   http://localhost:8080/AuctionApp/AuctionApp-app-client
   ```
4. You should observe Java Webstart loading the AuctionApp TestClient GUI.

   You should observe the following message output on the AuctionApp Log window of the GUI:

```
Received hello on startup
```

5. Using the Services tab of the IDE, verify that the application server has created 3 new rows in the AUCTIONUSER table.
6. Enter the following string in the AuctionApp TestClient GUI window.

```
hello
```

   You should see the following output:

```
User input: hello
Received client <-> server
```

7. Using the Services tab of the IDE, verify that AUCTIONUSER table contains only 3 rows. That is, the previous step did not add any new rows.

Exercise 4: Implementing the addAuction Use Case

This exercise contains the following sections:

- Task 1 – Updating the `AuctionManager` Session Bean
- Task 2 – Building, Deploying, and Testing the Application

Tool Reference – Tool references used in this exercise:

- Java EE Development: Enterprise Application Projects: Creating Enterprise Application Projects
- Java Development: Java Application Projects: Opening Projects
- Java EE Development: Enterprise Application Projects: Building Java EE Applications
- Java EE Development: Enterprise Application Projects: Verifying Java EE Applications
- Java EE Development: Enterprise Application Projects: Deploying Java EE Applications
- Java Development: Java Application Projects: Running Projects
- Server Resources: J2EE Application Servers: Examining Server Log Files
- Java Development: Modifying Project Libraries
- Java Development: Java Classes: Modifying Java Classes: Adding Fields
- Java Development: Java Classes: Opening Java Classes
- Java Development: Other Files: Opening Files
- Java EE Application Servers: Starting Application Servers
- Server Resources: Databases: Starting JavaDB
- Server Resources: Databases: Creating a Java DB Database
- Server Resources: Databases: Connecting to Databases
- Java EE Development: Persistence Modules: Creating Persistence Units

Task 1 – Updating the AuctionManager Session Bean

In this task, you update the `AuctionManager` session bean by implementing the `addAuction` method. Figure 4-8 illustrates the `addAuction` method of the `AuctionManagerBean` class and the `Auction` entity that it uses.

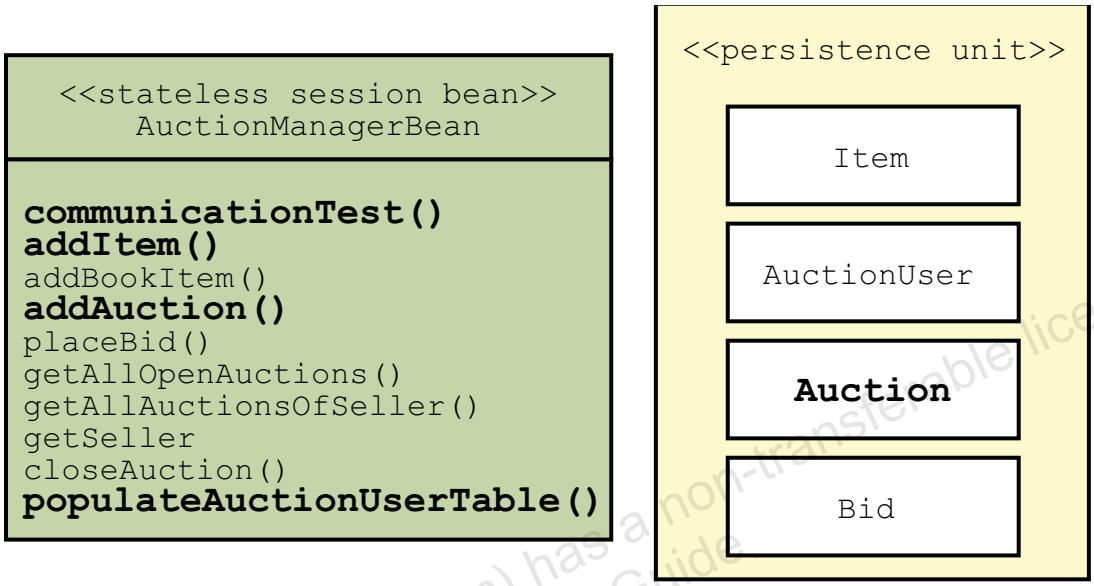

Figure 4-8 The `AuctionManagerBean` Class and Persistence Unit

Open the `AuctionManagerBean` class in a source editor window. Perform the following steps.

1. Import the following classes.

```
import auctionsystem.entity.Auction;
import auctionsystem.exception.*;
import javax.ejb.EJBException;
```

2. Type the `addAuction` method, as shown in Code 4-2.

Code 4-2 `addAuction` Method Template

```
1   public int addAuction(double startAmount, double increment,
2     Date closeTime,String itemDesc,String itemImage,Integer sellerID) {
3     try {
4       Item item = null;
5       AuctionUser seller = null;
6       Auction auction = null;
7       System.out.println("+++ aMan.addAuction: Creating new Item");
8       item = null; // action required
9       System.out.println("+++ aMan.addAuction: Created new Item");
10      seller = null; // action required
11      if (seller == null)
12        throw new AuctionException("Unknown seller ID " + sellerID);
13      System.out.println("+++ aMan.addAuction found seller " + seller);
14      auction = null; // action required
15       // action required;
16      System.out.println("+++ aMan.addAuction created auction " +
auction;
17      return auction.getId();
18    } catch (Exception ex) {
19      throw new EJBException(ex.getMessage());
20    }
21  }
```

3. Replace the null in code line 8 with a method invocation of the `addItem` method. You should pass `itemDesc` and `itemImage` as parameters to the `addItem` method.
4. Replace the null in code line 10 with a method invocation of the `find` method on the entity manager `em`. You are looking for the `AuctionUser` object that corresponds to the primary key value in the input parameter `sellerID`.
5. Replace the null in code line 14 with a call to the Auction entity constructor. In this line, you are creating a new Auction instance using `startAmount`, `increment`, `closeTime`, `item`, and `seller`.
6. Replace the comment in code line 15 with a method invocation of the `persist` method on the entity manager `em`. You are persisting the `auction` object you created in the previous line.
7. Build the `AuctionApp` project. Correct any errors you encounter.
8. Verify the `AuctionApp` project.
9. Deploy the `AuctionApp` application.

Task 2 – Building, Deploying, and Testing the Application

In this task, you test the `AuctionApp` application. In this test, you use the `TestClient` class to create one or more entries in the `AUCTION` table in the database.

Complete the following steps:

1. Use a browser to load the following URL:

   `http://localhost:8080/AuctionApp/AuctionApp-app-client`

2. You should observe Java Webstart loading the `AuctionApp TestClient` GUI.

   You should observe the following message output on the `AuctionApp Log` window of the GUI:

```
Received hello on startup
```

3. Enter the following string in the AuctionApp TestClient GUI window.

```
addAuction 5000 100 5 car car.jpg 2
```

   You should see the following output:

```
User input: addAuction 5000 100 5 car car.jpg 2
User #2 added an auction with the ID = 6 (Note: Actual ID might vary).
```

4. Examine the ITEM and AUCTION tables in the `Services` tab of the IDE.

   You should observe a new item corresponding to the car.

   You should observe a new auction corresponding to the auction you just created.

5. Repeat the previous two steps to add additional auctions. You can optionally test error cases, such as creating an auction with a non-existent auction user ID, for example an auction user ID of 20.

Exercise 5: Completing Review Questions

In this exercise, you answer questions about the material covered in Module 4 of the Student Guide.

Complete the following questions:

1. True or False: The Java Persistence API requires an application server?
2. What is the fully qualified annotation used by classes to be marked as Entity classes?
3. For an entity class, what is the alternative to field-based access?
4. Which annotation is required to specify the primary key field or property?

Exercise Summary

Discussion – Take a few minutes to discuss what experiences, issues, or discoveries you had during the lab exercise.

- Experiences
- Interpretations
- Conclusions
- Applications

Lab 5

Implementing Entity Classes: Modelling Data Association Relationships

Objectives

Upon completion of this lab, you should be able to:

- Create association relationships between the entity classes
- Implement the `placeBid` use case
- Complete the review questions

Exercise 1 – Creating Association Relationships Between the Entity Classes

This exercise contains the following sections:

- Task 1 – Creating the One-to-One Association Between `Auction` and `Item` Entities
- Task 2– Creating the Many-to-One/One-to-Many Association Between `Auction` and `AuctionUser` Entities
- Task 3 – Creating the Many-to-One/One-to-Many Association between `Bid` and `Auction` Entities
- Task 4 – Creating the Many-to-One/One-to-Many Association between `Bid` and `AuctionUser` Entities

Tool Reference – Tool references used in this exercise:

- Java EE Development: Enterprise Application Projects: Creating Enterprise Application Projects
- Java Development: Java Application Projects: Opening Projects
- Java EE Development: Enterprise Application Projects: Building Java EE Applications
- Java EE Development: Enterprise Application Projects: Verifying Java EE Applications
- Java EE Development: Enterprise Application Projects: Deploying Java EE Applications
- Java Development: Java Application Projects: Running Projects
- Server Resources: J2EE Application Servers: Examining Server Log Files
- Java Development: Modifying Project Libraries
- Java Development: Java Classes: Modifying Java Classes: Adding Fields
- Java Development: Java Classes: Opening Java Classes
- Java Development: Other Files: Opening Files
- Java EE Application Servers: Starting Application Servers
- Server Resources: Databases: Starting JavaDB
- Server Resources: Databases: Creating a Java DB Database
- Server Resources: Databases: Connecting to Databases

Task 1 – Creating the One-to-One Association Between Auction and Item Entities

In this task, you update the Auction entity class to implement the one-to-one unidirectional relationship between an auction and an item. Figure 5-1 illustrates the relationship between the Auction and the Item entities.

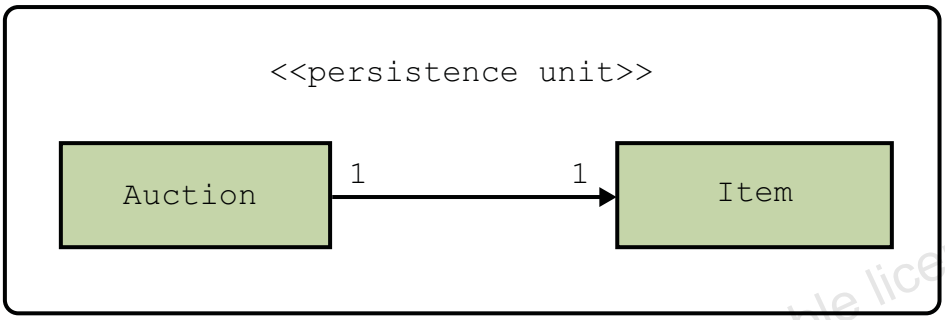

Figure 5-1 The Relationship Between the Auction and Item Entities

Open the Auction class in a source editor window. Perform the following steps.

1. Add the following field to the Auction class.

```
private Item item;
```

2. Use the editor's refactor feature to encapsulate (create getter and a setter methods) for the item field you added in the previous step.

   Annotate the getItem method you added in this step (by refactoring) with a OneToOne metadata annotation.

```
@OneToOne                    // add this line
private Item getItem() {     // code added by refatoring
```

   You need to import the OneToOne annotation.

```
import javax.persistence.OneToOne;
```

3. Locate the Auction constructor that takes multiple arguments.

   Add the following line of code to the constructor.

```
setItem(item);
```

4. Build the AuctionApp project. Correct any errors you encounter.
5. Verify the AuctionApp project.
6. Deploy the AuctionApp application.
7. Use the Services tab of the IDE to examine the AUCTION Table.

   Right click on the AUCTION Table node and select the View Data menu option. You should observe a new join column ITEM_ID.

Task 2– Creating the Many-to-One/One-to-Many Association Between Auction and AuctionUser Entities

In this task, you update the `Auction` and `AuctionUser` entity classes to implement the many-to-one/one-to-many bidirectional relationship between the two entity classes. In this exercise, you edit both the `Auction` and `AuctionUser` classes. Figure 5-2 illustrates the relationship between the `Auction` and the `AuctionUser` entities.

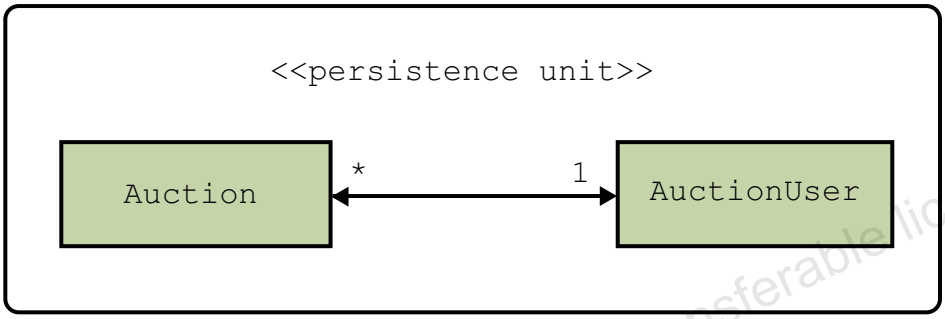

Figure 5-2 The Relationship Between the `Auction` and `AuctionUser` Entities

Open the `Auction` class in a source editor window. Perform the following steps.

1. Add the following field to the `Auction` class.

```
private AuctionUser seller;
```

2. Use the editor's refactor feature to encapsulate (create getter and a setter methods) for the `seller` field you added in the previous step.

   Annotate the `getSeller` method you added in this step (by refactoring) with a `ManyToOne` metadata annotation.

```
@ManyToOne                          // add this line
private AuctionUser getSeller() { // code added by refatoring
```

   You need to import the `ManyToOne` annotation.

```
import javax.persistence.ManyToOne;
```

3. Locate the `Auction` constructor that takes multiple arguments.

   Add the following line of code to the `Auction` constructor.

```
setSeller(seller);
```

4. Build the `AuctionApp` project. Correct any errors you encounter.

Open the `AuctionUser` class in a source editor window. Perform the following steps.

1. Add the following field to the `AuctionUser` class.

```
private Collection <Auction> auctions;
```

Remember to import the `java.util.Collection` class.

2. Use the editor's refactor feature to encapsulate (create getter and a setter methods) for the `auctions` field you added in the previous step.

Annotate the `getAuctions` method you added in this step (by refactoring) with a `OneToMany` metadata annotation.

```
@OneToMany                          // add this line
private Collection <Auction> getAuctions() { // code added by refatoring
```

You need to import the `OneToMany` annotation.

```
import javax.persistence.OneToMany;
```

3. Add the following method to the AuctionUser class.

```
public void addAuction(Auction newAuction) {
  try {
    auctions.add(newAuction);
  }catch (UnsupportedOperationException uoe){
     System.out.println("AuctionUser.addAuction " + uoe.getMessage());
  }
}
```

4. Build the `AuctionApp` project. Correct any errors you encounter.
5. Verify the `AuctionApp` project.
6. Deploy the `AuctionApp` application.
7. Use the `Services` tab of the IDE to examine the `AUCTION` Table.

Right click on the `AUCTION` Table node and select the View Data menu option. You should observe a new join column `SELLER_ID`.

Discussion – What would be the impact of omitting the `mappedBy` attribute shown in Step 2?

Figure 5-3 demonstrates the impact of using the `mappedBy` attribute. The `mappedBy` attribute creates a bidirectional relationship between an `Auction` instance and a `Seller` instance.

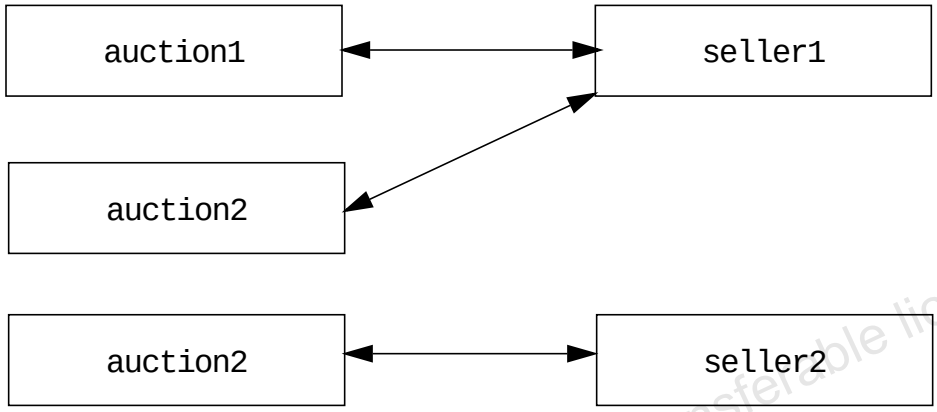

Figure 5-3 Relationships Between Sellers and Auctions Using the `mappedBy` Attribute

Figure 5-4 demonstrates the impact of not using the `mappedBy` attribute. The absence of the `mappedBy` attribute creates two unidirectional relationships between an `Auction` instance and a `Seller` instance.

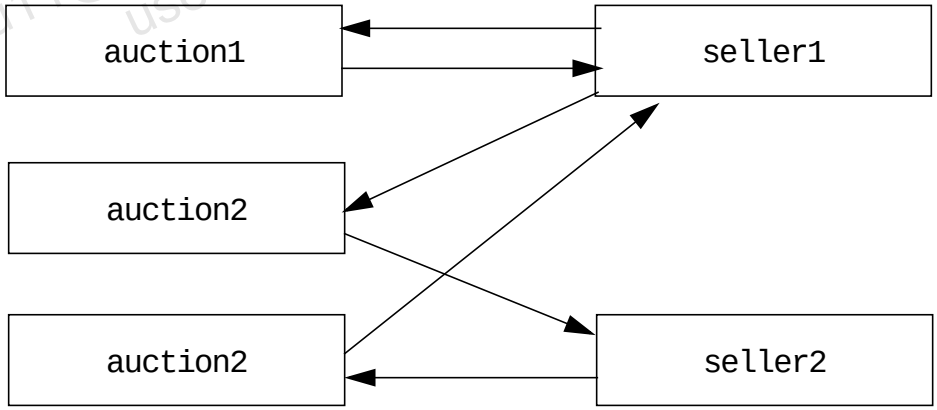

Figure 5-4 Relationships Between Sellers and Auctions Not Using the `mappedBy` Attribute

Task 3 – Creating the Many-to-One/One-to-Many Association between Bid and Auction Entities

In this task, you update the Bid and Auction entity classes to implement the many-to-one/one-to-many bidirectional relationship between the two entity classes. In this exercise you edit both the Bid and Auction classes. Figure 5-5 illustrates the relationship between the Auction and the Bid entities.

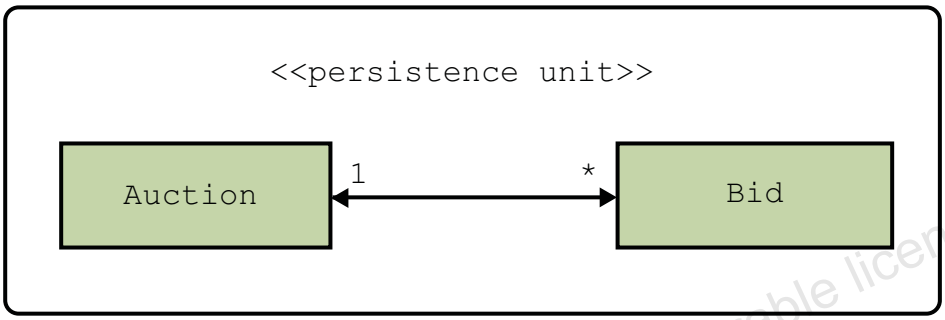

Figure 5-5 The Relationship Between the Auction and Bid Entities

Perform the following steps.

1. Use the information provided in Table 5-1 to add the relationship fields to the Bid and Auction classes.

Table 5-1 Bid – Auction Association Relationship Fields

Class	Annotation	Field Type	Field Name
Bid	@ManyToOne	Auction	auction
Auction	@OneToMany	Collection <Bid>	bids

For each relationship field you add, ensure that you also create corresponding encapsulation (getter and setter) methods.

Note – The relationship annotations must be added to the getter methods.

2. Add the following method to the Auction class.

```
public void addBid(Bid newBid) {
  try {
    bids.add(newBid);
  }catch (UnsupportedOperationException uoe){
     // TBD log exception
  }
}
```

3. Build the `AuctionApp` project. Correct any errors you encounter.
4. Verify the `AuctionApp` project.
5. Deploy the `AuctionApp` application.
6. Use the `Services` tab of the IDE to the `BID` table.

   Right click on the `BID` Table node and select the View Data menu option. You should observe a new join column `AUCTION_ID`.

Task 4 – Creating the Many-to-One/One-to-Many Association between Bid and AuctionUser Entities

In this task, you update the Bid and AuctionUser entity classes to implement the many-to-one/one-to -many bidirectional relationship between the two entity classes. In this exercise, you edit both the Bid and AuctionUser classes. Figure 5-6 illustrates the relationship between the AuctionUser and the Bid entities.

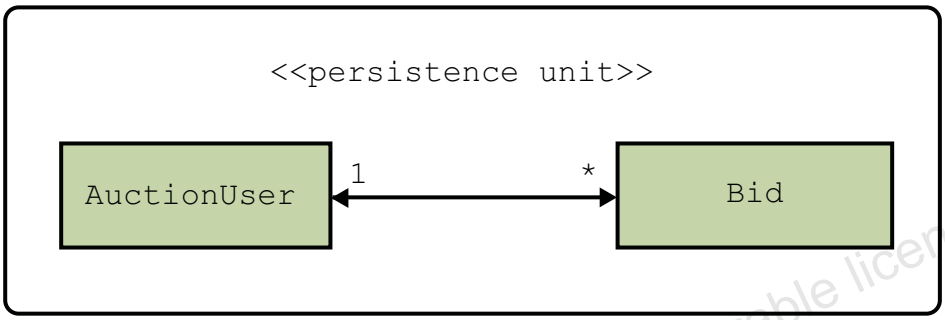

Figure 5-6 The Relationship Between the AuctionUser and Bid Entities

Perform the following steps.

1. Use the information provided in Table 5-2 to add the relationship fields to the Bid and AuctionUser classes.

Table 5-2 Bid – AuctionUser Association Relationship Fields

Class	Annotation	Field Type	Field Name
Bid	@ManyToOne	AuctionUser	bidder
AuctionUser	@OneToMany	Collection <Bid>	bids

For each relationship field you add, ensure that you also create corresponding encapsulation (getter and setter) methods.

2. Add the following method to the AuctionUser class.

```
public void addBid(Bid newBid) {
  try {
    bids.add(newBid);
  }catch (UnsupportedOperationException uoe){
     // TBD log exception
  }
}
```

3. Build the AuctionApp project. Correct any errors you encounter.

4. Verify the `AuctionApp` project.
5. Deploy the `AuctionApp` application.
6. Use the `Services` tab of the IDE to the `BID` table.

   Right click on the `BID` Table node and select the View Data menu option. You should observe a new join column `BIDDER_ID`.

Exercise 2 – Implementing the placeBid use Case: Phase 1

This exercise contains the following sections:

- Task 1– Implementing the placeBid Method (Phase 1)
- Task 2 – Testing the placeBid Method (Phase 2)

Tool Reference – Tool references used in this exercise:

- Java EE Development: Enterprise Application Projects: Creating Enterprise Application Projects
- Java Development: Java Application Projects: Opening Projects
- Java EE Development: Enterprise Application Projects: Building Java EE Applications
- Java EE Development: Enterprise Application Projects: Verifying Java EE Applications
- Java EE Development: Enterprise Application Projects: Deploying Java EE Applications
- Java Development: Java Application Projects: Running Projects
- Server Resources: J2EE Application Servers: Examining Server Log Files
- Java Development: Modifying Project Libraries
- Java Development: Java Classes: Modifying Java Classes: Adding Fields
- Java Development: Java Classes: Opening Java Classes
- Java Development: Other Files: Opening Files
- Java EE Application Servers: Starting Application Servers
- Server Resources: Databases: Starting JavaDB
- Server Resources: Databases: Creating a Java DB Database
- Server Resources: Databases: Connecting to Databases

Task 1– Implementing the placeBid Method (Phase 1)

In this task, you update the AuctionManager session bean by implementing the placeBid method. The objective of the phase 1 implementation of the placeBid method is to create a Bid entry in the database from the information provided in the input parameters. In creating the Bid entry, you do the following:

- Ignore all business rules that govern the creation of a bid. For example, you ignore the rules, such as the bidder cannot be the seller or the new bid must be greater than the current greatest bid.
- Do not check the bidder's credit standing. Instead, you approve all bids.
- Do not establish the relationship between the newly created bid and the auction or between the bid and the auction user.

You complete these tasks in a subsequent exercise.

Figure 5-7 illustrates the placeBid method of the AuctionManagerBean class and the Bid entity that it uses.

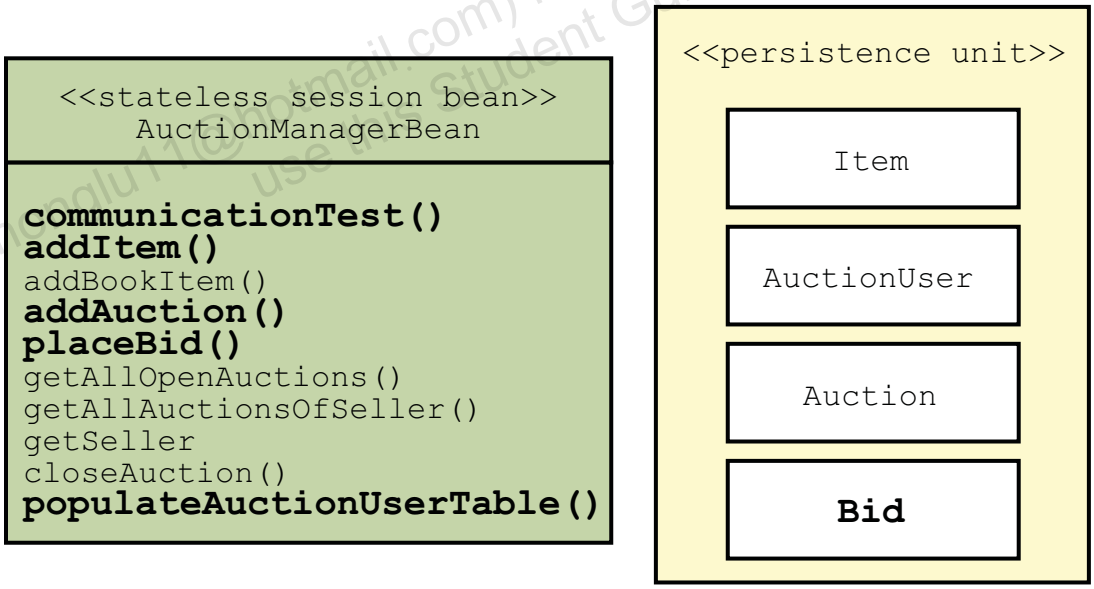

Figure 5-7 The placeBid Method of the AuctionManagerBean Class

Open the AuctionManagerBean class in a source editor window. Perform the following steps.

1. Import the following class:

```
import auctionsystem.entity.Bid;
```

2. Type the placeBid method, as shown in the code listing Code 5-1 on page Lab 5-14. Use the following hints to supply the missing code.

   Hint 1: Use entity manager em to find the auction corresponding to the input parameter auctionID.

   Hint 2: Find the bid corresponding to the input parameter bidderID.

   Hint 3: Your objective in line15 of Code 5-1, is to create a new Bid object using the following constructor:

```
public Bid(double amount, String authorization, Auction auction,
AuctionUser bidder)
```

   To invoke the constructor, you need values for the following parameters:

   a. amount

      You can use the input argument bidAmount for the amount parameter.

   b. authorization

      For the phase 1 implementation, you use the local variable authorization declared in line 10 of Code 5-1 on page Lab 5-14.

   c. auction

      Use the local variable auction declared in line 3 of Code 5-1 on page Lab 5-14. However, you first need to find, from the database, and assign to auction, the Auction instance with the primary key value held in the input argument auctionID.

   d. bidder

      Use the local variable bidder declared in 4 of Code 5-1 on page Lab 5-14. However, you first need to find from the database and assign to bidder the AuctionUser instance with primary key value held in the input argument bidderID.

   Hint 4: Use entity manager em to persist the newBid object.

Code 5-1 placeBid Method Template

```
1   public void placeBid(Integer auctionID, Integer bidderID,
2     double bidAmount) throws AuctionException {
3        Auction auction = /* your code here as per hint 1 */
4        AuctionUser bidder = /* your code here as per hint 2 */
5
6       System.out.println("AMB.checkBid(auction, bidder, bidAmount "
7         + auction.getId() + "  " + bidder.getId() + "  " + bidAmount);
8
9       // Get the payment authorization:
10      String authorization = “Approved”; // for phase 1
11      if (authorization.compareTo("DENIED") == 0) {
12        throw new CreditCardException("Authorization denied");
13      }
14
15      Bid newBid = /* your code here as per hint 3 */
16      /* Your code here as per hint 4 */
17  }
```

3. Build the AuctionApp project. Correct any errors you encounter.
4. Verify the AuctionApp project.

Task 2 – Testing the placeBid Method (Phase 1)

In this task, you test the placeBid method.

1. Deploy the AuctionApp application.
2. Use a browser to load the following URL:

   http://localhost:8080/AuctionApp/AuctionApp-app-client
3. You should observe Java Webstart loading the AuctionApp TestClient GUI.

   You should observe the following message output on the AuctionApp Log window of the GUI:

```
Received hello on startup
```

4. Use the Services tab of the IDE to verify that the application server has been created to view the data in the AUCTIONUSER table. It should be populated.
5. Enter the following string in the AuctionApp TestClient GUI window.

```
addAuction 5000 100 5 car car.jpg 3
```

   You should see the following output:

```
User input: addAuction 5000 100 5 car car.jpg 3
User #3 added an auction with the ID = 6 (Note: Actual ID might vary).
```

6. Use the Services tab of the IDE to verify that the application server has been created to view the data in the AUCTION and ITEM tables. They should contain an entry for the auction and item you created.
7. Enter the following string in the AuctionApp TestClient GUI window.

```
placeBid 6 2 5500
```

   You should see the following output:

```
User input: placeBid 6 2 5500
User #2 has placed a bid
```

8. Use the Services tab of the IDE to examine the BID table. It should contain an entry for the bid you created.
9. Repeat the last two steps with various values for placeBid. Test the error cases, such as placing a low bid or the seller bidding on the auction.For example, try the following:

   Place a low bid by current highest bidder:

```
placeBid 6 2 3000
```

Place a bid by a different bidder:

```
placeBid 6 4 2300
```

Place a bid by seller:

```
placeBid 6 3 1000
```

Use the Services tab of the IDE to examine the BID table. It should contain an entry for all the bids you created, even those that broke the business rules.

Also note that the columns named AUCTION_ID and BIDDER_ID are null.

Exercise 3 – Implementing the placeBid Use Case: Phase 2

This exercise contains the following sections:

- Task 1 – Implementing the placeBid Method (Phase 2)
- Task 2 – Testing the placeBid Method (Phase 2)

Tool Reference – Tool references used in this exercise:

- Java EE Development: Enterprise Application Projects: Creating Enterprise Application Projects
- Java Development: Java Application Projects: Opening Projects
- Java EE Development: Enterprise Application Projects: Building Java EE Applications
- Java EE Development: Enterprise Application Projects: Verifying Java EE Applications
- Java EE Development: Enterprise Application Projects: Deploying Java EE Applications
- Java Development: Java Application Projects: Running Projects
- Server Resources: J2EE Application Servers: Examining Server Log Files
- Java Development: Modifying Project Libraries
- Java Development: Java Classes: Modifying Java Classes: Adding Fields
- Java Development: Java Classes: Opening Java Classes
- Java Development: Other Files: Opening Files
- Java EE Application Servers: Starting Application Servers
- Server Resources: Databases: Starting JavaDB
- Server Resources: Databases: Creating a Java DB Database
- Server Resources: Databases: Connecting to Databases

Task 1 – Implementing the placeBid Method (Phase 2)

In this task, you continue the update of the placeBid method of the AuctionManager session bean. The primary objective of phase 2 is to:

- Establish the one-to-many/many-to-one relationship between the Auction and Bid entities.
- Establish the one-to-many/many-to-one relationship between the AuctionUser and Bid entities.

In "Exercise 1 – Creating Association Relationships Between the Entity Classes" on page Lab 5-2, you created the static relationship between the entities. In this exercise, for every bid you create, you update the corresponding Auction and AuctionUser user entities to reflect their relationship with the Bid entity. Figure 5-7 illustrates the relationship between the Bid entity and the Auction and the AuctionManager entities.

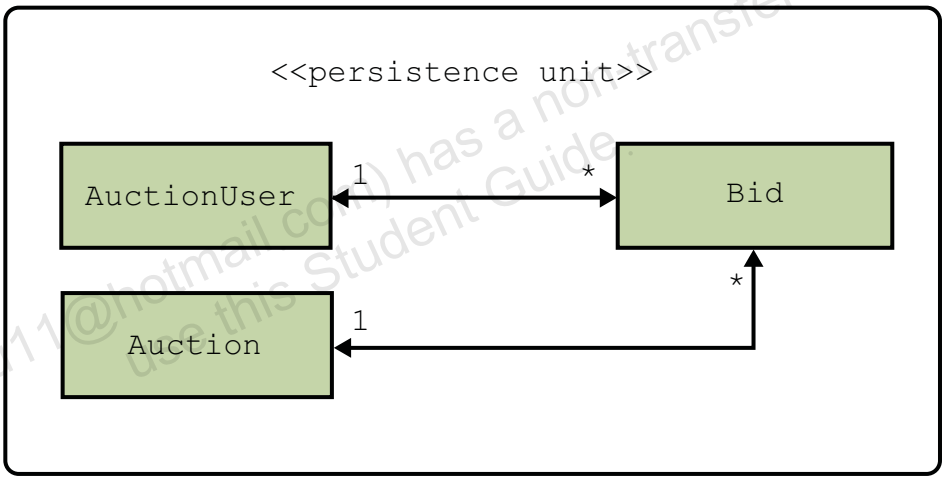

Figure 5-8 The Bid to Auction and AuctionUser Relationships

To update the Auction entity, perform the following steps.

1. Open the Auction class in a source editor window.
2. Identify the method that takes a single Bid object and adds it to the relationship field bids collection. Hint: The method begins with add.
3. Open the AuctionManagerBean class in a source editor window and examine the end of the placeBid method.

   To add the newBid instance to the auction object, invoke the method you identified in the previous step on the auction object and pass it the newBid instance.

   Remember to merge the auction object.

To update the AuctionUser entity, complete the following steps.

1. Open the AuctionUser class in a source editor window.
2. Identify the method that takes a single Bid object and adds it to the relationship field bids collection. Hint: The method begins with add.
3. Open the AuctionManagerBean class in a source editor window and examine the end of the placeBid method.

   To add the newBid instance to the bidder object, invoke the method you identified in the previous step on the bidder object and pass it the newBid instance.

   Remember to merge the bidder object.

To update the Bid entity, complete the following steps.

1. Open the Bid class in a source editor window.
2. Examine the body of the following constructor.

```
public Bid(double amount, String authorization, Auction auction,
AuctionUser bidder)
```

   Add the following code lines to the constructor.

```
setAuction(auction);
setBidder(bidder);
```

3. Build the AuctionApp project. Correct any errors you encounter.
4. Verify the AuctionApp project.

Task 2 – Testing the placeBid Method (Phase 2)

In this task, you update the AuctionManager session bean by implementing the addAuction method.

1. Deploy the AuctionApp application.
2. Use the Services tab of the IDE to verify that the application server has created the AUCTION, AUCTIONUSER, BID, and ITEM tables.

   Verify that each table is empty.
3. Use a browser to load the following URL:

   `http://localhost:8080/AuctionApp/AuctionApp-app-client`
4. You should observe Java Webstart loading the AuctionApp TestClient GUI.

   You should observe the following message output on the AuctionApp Log window of the GUI:

```
Received hello on startup
```

5. Use the Services tab of the IDE to verify that the application server has been created to view the data in the AUCTIONUSER table. It should be populated.
6. Enter the following string in the AuctionApp TestClient GUI window.

```
addAuction 5000 100 5 car car.jpg 3
```

   You should see the following output:

```
User input: addAuction 5000 100 5 car car.jpg 3
User #3 added an auction with the ID = 6(Note: Actual ID might vary).
```

7. Use the Services tab of the IDE to verify that the application server has been created to view the data in the AUCTION and ITEM tables. They should contain an entry for the auction and item you created.
8. Enter the following string in the AuctionApp TestClient GUI window.

```
placeBid 6 2 5500
```

   You should see the following output:

```
User input: placeBid 6 2 5500
User #2 has placed a bid
```

9. Use the Services tab of the IDE to verify that the application server has been created to view the data in the BID table. It should contain an entry for the bid you created.

   Also note that the columns named AUCTION_ID and BIDDER_ID are no longer null.

10. Repeat the last two steps with various values for placeBid. Test the error cases, such as placing a low bid or the seller bidding on the auction. For example, try the following:

    Place a low bid by current highest bidder:

```
placeBid 6 2 6000
```

    Place a bid by a different bidder:

```
placeBid 6 4 2300
```

    Place a bid by seller:

```
placeBid 6 3 1000
```

    Use the Services tab of the IDE to the data in the BID table. It should contain an entry for all the bids you created even those that broke the business rules.

Exercise 4 – Implementing the placeBid Use Case: Phase 3 (Optional)

This exercise contains the following sections:

- Task 1 – Copying the Helper Class
- Task 2 – Implementing the placeBid Method (Phase 3)
- Task 3– Testing the placeBid Method (Phase 3)

This exercise implements the business rules that govern the acceptance of new bids on an auction. You have the option to skip this exercise, because none of the following exercises depend on the completion of this exercise.

Tool Reference – Tool references used in this exercise:

- Java EE Development: Enterprise Application Projects: Creating Enterprise Application Projects
- Java Development: Java Application Projects: Opening Projects
- Java EE Development: Enterprise Application Projects: Building Java EE Applications
- Java EE Development: Enterprise Application Projects: Verifying Java EE Applications
- Java EE Development: Enterprise Application Projects: Deploying Java EE Applications
- Java Development: Java Application Projects: Running Projects
- Server Resources: J2EE Application Servers: Examining Server Log Files
- Java Development: Modifying Project Libraries
- Java Development: Java Classes: Modifying Java Classes: Adding Fields
- Java Development: Java Classes: Opening Java Classes
- Java Development: Other Files: Opening Files
- Java EE Application Servers: Starting Application Servers
- Server Resources: Databases: Starting JavaDB
- Server Resources: Databases: Creating a Java DB Database
- Server Resources: Databases: Connecting to Databases

Task 1 – Copying the Helper Class

Complete the following steps:.

1. Open the Favorites tab and locate the following directory:

```
exercises/mod05_entity2/exercise4
```

Copy the following file:

- `AuctionHelper.java`

2. Use the Files tab to paste the files in the following directory:

```
AuctionApp-ejb/src/java/auctionsystem/helper
```

3. Build the `AuctionApp-ejb` project. Correct any errors you encounter.
4. Open the `AuctionHelper` class in a source editor window.

The following information is provided to assist you with understanding the services provided by the `AuctionHelper` class. The `AuctionHelper` class contains a number of methods that encapsulate business rules needed by the `placeBid` use case. The list provides an introduction to these methods:

```
public String getBidAuthorization(double bidAmount, Integer bidderID)
```

This method always returns the string "`Approved`". A proper implementation of this method would use business rules to validate the bidders credit status and return `Approved` or `Denied`.

```
public Bid checkBid(Auction auction, AuctionUser bidder,
    double newAmount) throws AuctionException
```

This method either throws an exception or returns the current highest bid or null if no bids exist on the auction. An exception is thrown if any of the following business rules are not met:

- The auction must exist and must have an open status.
- The bidder must exist and the bidder must not be the seller or the holder of the current highest bid on the auction.
- The bid amount must be greater than the current greatest bid plus the bid increment.

```
public Bid getHighestBid(Auction auction)
```

This is a helper method for the `checkBid` method. It returns the current greatest bid. You will probably not need to invoke this method.

```
public void informAffectedBidders(Auction auction, Bid formerHighestBid,
Bid newBid)
```

This method sends courtesy notifications to the new highest bidder and previous highest bidder.

```
public void sendNotification(String emailAddress, String message)
```

This is a helper method for the `informAffectedBidders` method. The current implementation prints to the terminal.

Task 2 – Implementing the placeBid Method (Phase 3)

In this task, you continue the update of the `placeBid` method of the `AuctionManager` session bean. The primary objective of phase 3 is to:

- Implement all business rules that govern the creation of a bid. The `placeBid` method must throw the `AuctionException` if any of the following statements is true.
  - The bidder is the seller.
  - If the new bid is less than the current highest bid plus the increment value specified in the auction.
  - If the new bid is from the current highest bidder.
- Make provision in the code for a future capability to check the bidders credit standing.
- Optionally, notify the bidder of success or failure and notify the previous highest bidder of a an out-bid event.

You are free to implement Phase 3 any way you like. You can use the following suggestions.

To implement the business rules, complete the following steps:

1. Open the `AuctionManagerBean` class in a source editor window and examine the end of the `placeBid` method.
2. Import the following classes:

```
import auctionsystem.helper.AuctionHelper;
```

3. Add the following instance variable:

```
private AuctionHelper auctionHelper = new AuctionHelper();
```

Figure 5-9 illustrates the composition relationship between the AuctionManagerBean class and the AuctionHelper class.

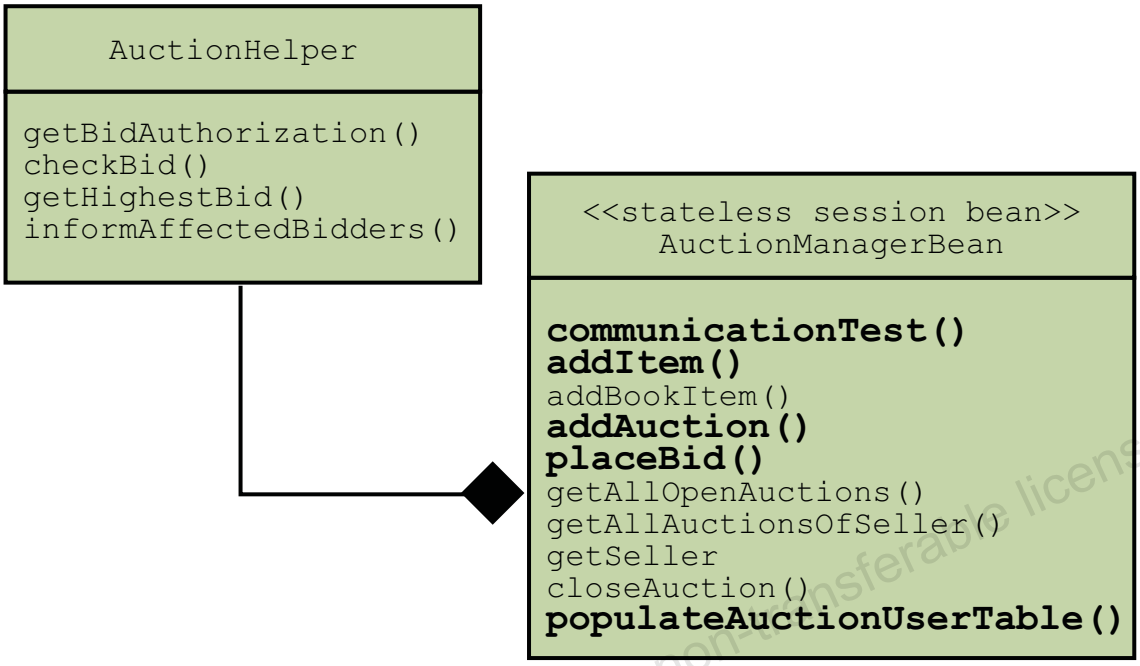

Figure 5-9 The AuctionHelper Class

4. Examine the placeBid method.

   Type the following line of code before the newBid instance creation.

```
Bid previousHigestBid = auctionHelper.checkBid(auction, bidder,
bidAmount);
```

5. To implement the provision in the code for a future capability to check the bidders credit standing, perform the following steps:

   Examine the placeBid method and locate the line:

```
if (authorization.compareTo("DENIED") == 0) {
```

   Type the following line of code *prior* to the previously identified line:

```
String authorization = auctionHelper.getBidAuthorization(bidAmount,
  bidderID);
```

6. To optionally notify the bidder of success or failure and notify the previous highest bidder of an out bid event, complete the following steps:

   a. Open the AuctionManagerBean class in a source editor window and examine the end of the placeBid method.

b. Type the following line of code after the newBid instance creation.

```
auctionHelper.informAffectedBidders(auction, previousHighestBid, newBid);
```

c. Build the AuctionApp project. Correct any errors you encounter.

d. Verify the AuctionApp project.

Task 3– Testing the placeBid Method (Phase 3)

In this task, you update the AuctionManager session bean by implementing the addAuction method.

1. Deploy the AuctionApp application.
2. Use the Services tab of the IDE to verify that the application server has created the AUCTION, AUCTIONUSER, BID, and ITEM tables.

   Verify that each table is empty.
3. Use a browser to load the following URL:

   `http://localhost:8080/AuctionApp/AuctionApp-app-client`
4. You should observe Java Webstart loading the AuctionApp TestClient GUI.

   You should observe the following message output on the AuctionApp Log window of the GUI:

```
Received hello on startup
```

5. Use the Services tab of the IDE to verify that the application server has been created to view the data in the AUCTIONUSER table. It should be populated.
6. Enter the following string in the AuctionApp TestClient GUI window.

```
addAuction 5000 100 5 car car.jpg 3
```

   You should see the following output:

```
User input: addAuction 5000 100 5 car car.jpg 3
User #3 added an auction with the ID = 6(Note: Actual ID might vary).
```

7. Use the Services tab of the IDE to verify that the application server has been created to view the data in the AUCTION and ITEM tables. They should contain an entry for the auction and item you created.
8. Enter the following string in the AuctionApp TestClient GUI window.

```
placeBid 6 2 5500
```

You should see the following output:

```
User input: placeBid 6 2 5500
User #2 has placed a bid
```

9. Use the Services tab of the IDE to verify that the application server has been created to view the data in the BID table. It should contain an entry for the bid you created.

10. Repeat the last two steps with various values for placeBid. Test the error cases, such as placing a low bid or the seller bidding on the auction.

Exercise 5: Completing Review Questions

In this exercise, you answer questions about the material covered in Module 5 of the Student Guide.

Complete the following questions:

1. Which annotation is used to specify a one-to-one relationship?
2. In many-to-one/one-to-many associations, which side cannot be the owning entity?

Exercise Summary

Discussion – Take a few minutes to discuss what experiences, issues, or discoveries you had during the lab exercise.

- Experiences
- Interpretations
- Conclusions
- Applications

Lab 6

Implementing Entity Classes: Modelling Inheritance Relationships

Objectives

Upon completion of this lab, you should be able to:

- Define a parent entity class
- Define a child entity class
- Complete the review questions

Exercise 1 – Defining the Parent Entity Class (Optional)

This exercise contains the following sections:

- Task 1 – Updating the `Item` Entity Class (Overview)
- Task 2 – Updating the `Item` Entity Class (Detail)

You have the option to skip this exercise. If you do, you must also skip the following exercise "Exercise 2 – Defining the Child Entity Class (Optional)" on page 6-5. None of the exercises in the following modules depend on the completion of any exercise in this module.

Tool Reference – Tool references used in this exercise:

- Java EE Development: Enterprise Application Projects: Creating Enterprise Application Projects
- Java Development: Java Application Projects: Opening Projects
- Java EE Development: Enterprise Application Projects: Building Java EE Applications
- Java EE Development: Enterprise Application Projects: Verifying Java EE Applications
- Java EE Development: Enterprise Application Projects: Deploying Java EE Applications
- Java Development: Java Application Projects: Running Projects
- Server Resources: J2EE Application Servers: Examining Server Log Files
- Java Development: Modifying Project Libraries
- Java Development: Java Classes: Modifying Java Classes: Adding Fields
- Java Development: Java Classes: Opening Java Classes
- Java Development: Other Files: Opening Files
- Java EE Application Servers: Starting Application Servers
- Server Resources: Databases: Starting JavaDB
- Server Resources: Databases: Creating a Java DB Database
- Server Resources: Databases: Connecting to Databases

Task 1 – Updating the `Item` Entity Class (Overview)

In this task, you update the `Item` entity class to be the parent class of an entity inheritance hierarchy.

This task requires that you specify the inheritance strategy to be used by the `Item` entity and its subclasses. The `Item` entity uses the joined subclass inheritance strategy. Table 6-1 contains the information you require to update the `Item` class.

Table 6-1 Inheritance Strategy Information

Annotation	Value	Comment

`Inheritance(`
`  strategy`
`)`
	`InheritanceType.JOINED`	Specifies the inheritance strategy

`DiscriminatorColumn(`
`  name,`
`  discriminatorType,`
`  length`
`)`
	`"DISC"`
	`DiscriminatorType.STRING`
	20
		Specifies the discriminator column

`DiscriminatorValue`	`"Item"`	Specifies the discriminator value to be used for the `Item` entity

If you need more detailed instructions, proceed to "Task 2 – Updating the `Item` Entity Class (Detail)" on page Lab 6-4.

The following instructions are for students who require minimal information to complete the task.

Complete the following steps.

1. Use the information provided in Table 6-1 to update the Item class.
2. Build the `AuctionApp` project. Correct any errors you encounter.
3. Verify the `AuctionApp` project.
4. Deploy the `AuctionApp` application.
5. Use the `Services` tab of the IDE to examine the `ITEM` table.
6. Proceed to Exercise 2.

Task 2 – Updating the `Item` Entity Class (Detail)

The following instructions are for students who require detailed information to complete the task of updating the Item entity class.

Open the `Item` entity class in a source editor window. Perform the following steps.

1. Import the following annotations:

```
import javax.persistence.Inheritance;
import javax.persistence.DiscriminatorColumn;
import javax.persistence.InheritanceType;
import javax.persistence.DiscriminatorType;
import javax.persistence.DiscriminatorValue;
```

2. Insert the following lines of code between the `Entity` annotation and the `class` keyword:

```
@Inheritance(strategy=InheritanceType.JOINED)
@DiscriminatorColumn(name="DISC",
  discriminatorType=DiscriminatorType.STRING, length=20)
@DiscriminatorValue("Item")
```

3. Build the `AuctionApp` project. Correct any errors you encounter.
4. Verify the `AuctionApp` project.
5. Deploy the `AuctionApp` application.
6. Use the `Services` tab of the IDE to examine the `ITEM` table. At this stage, it might contain the discriminator column `DISC`. If not the discriminator column is created after the first inheriting entity class (in this case `BookItem`) is deployed.

Note – The timing of the creation of the discriminator column depends on the persistence provider implementation.

Exercise 2 – Defining the Child Entity Class (Optional)

This exercise contains the following sections:

- Task 1 – Creating the `BookItem` Entity Class
- Task 2 – Updating the `AuctionManager` Session Bean
- Task 3 – Testing the Creation of `Item` and `BookItem` Entries

This exercise depends on the successful completion of the previous exercise, "Exercise 1 – Defining the Parent Entity Class (Optional)" on page 6-2.

You have the option to skip this exercise. None of the exercises in the following modules depend on the completion of any exercise in this module.

Tool Reference – Tool references used in this exercise:

- Java EE Development: Enterprise Application Projects: Creating Enterprise Application Projects
- Java Development: Java Application Projects: Opening Projects
- Java EE Development: Enterprise Application Projects: Building Java EE Applications
- Java EE Development: Enterprise Application Projects: Verifying Java EE Applications
- Java EE Development: Enterprise Application Projects: Deploying Java EE Applications
- Java Development: Java Application Projects: Running Projects
- Java Development: Modifying Project Libraries
- Java Development: Java Classes: Modifying Java Classes: Adding Fields
- Java Development: Java Classes: Opening Java Classes
- Java Development: Other Files: Opening Files
- Java EE Application Servers: Starting Application Servers
- Server Resources: Databases: Starting JavaDB
- Server Resources: Databases: Creating a Java DB Database
- Server Resources: Databases: Connecting to Databases

Task 1 – Creating the BookItem Entity Class

In this task, you create the `BookItem` entity class as the child class of the `Item` entity class. Figure 6-1 illustrates the relationship between BookItem and `Item` and entities.

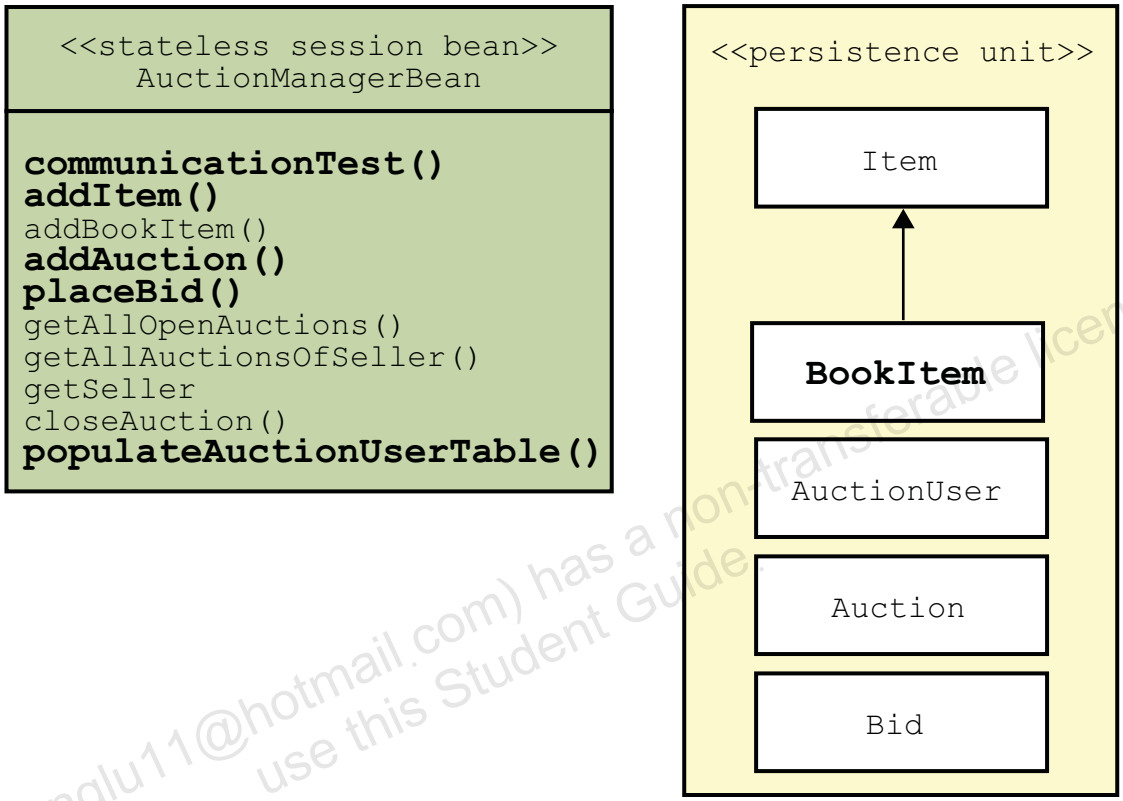

Figure 6-1 The `BookItem` Entity

Complete the following steps:

1. Open a New Entity Class dialog box.

   Right-click the `AuctionApp-ejb` project node and choose New > Entity Class

2. Enter the following information in the New Entity Class dialog box:

   Class Name: **`BookItem`**

   Project: **`Source Packages`**

   Package: **`auctionsystem.entity`**

   Primary key Type: **`Integer`**

3. Use the source editor to modify the signature of the `BookItem` to inherit from the `Item` class, in addition to implementing the `Serializable` interface. The class signature of the `BookItem` should resemble the following line of code:

```
public class BookItem extends Item implements Serializable {
```

4. Because the primary key for the `BookItem` class is inherited from the `Item`, you need to delete the following fields and methods:

   a. The serial version id and primary key fields

```
  private static final long serialVersionUID = 1L;
  private Integer id;
```

   b. The annotated `getId` method

```
  @Id
  @GeneratedValue(strategy = GenerationType.AUTO)
  public Integer getId () {
    return id;
  }
```

   c. Also, remove the corresponding `getId`, `setId`, `hashcode`, `equals`, and `toString` methods.

5. Modify the `toString` method so it will compile. You need to invoke the parent class's `getId` method.

6. Add the following fields to the `BookItem` class:

```
protected String title;
protected String author;
```

7. Encapsulate the fields you added in the previous step with public getter and setter methods.

   Right-click in the Source Editor and choose Refactor > Encapsulate fields to generate getters and setters for each of the fields. In the Encapsulate Fields dialog box, make sure that the getter and setter check boxes are selected for all of the fields

8. Add the following constructors to the `BookItem` class.

```
public BookItem () {
}

public BookItem (String description, String image, String title,
  String author) {
  super(description, image);
  setTitle(title);
  setAuthor(author);
}
```

9. Build the `AuctionApp` project. Correct any errors.
10. Verify the `AuctionApp` project.
11. Deploy the `AuctionApp` application.
12. Use the `Services` tab of the IDE to examine the `ITEM` table. Verify that the `ITEM` table contains a new column named DISC. Also verify that a new `BOOKITEM` table is created containing the columns `ID`, `TITLE`, and `AUTHOR`.

Task 2 – Updating the AuctionManager Session Bean

In this task, you update the AuctionManager session bean by implementing the addBookItem method. Figure 6-1 illustrates the addBookItem method of the AuctionManagerBean class.

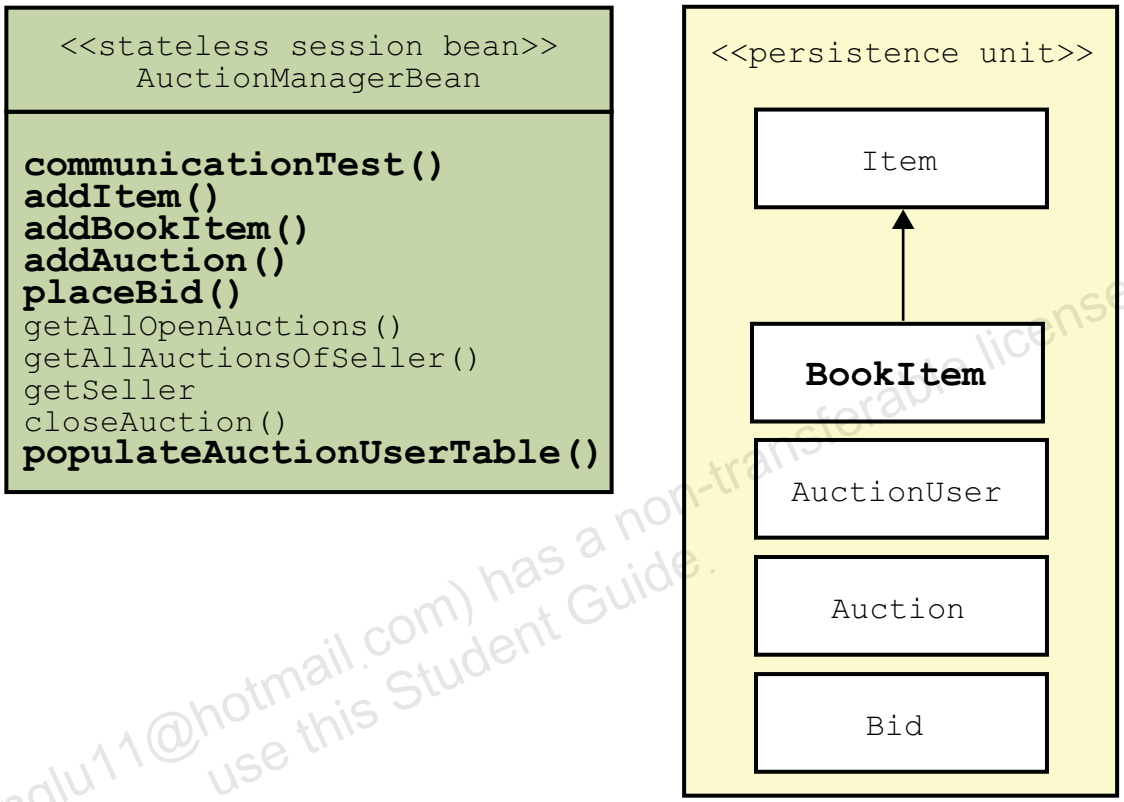

Figure 6-2 The addBookItem Method

Complete the following steps:

1. Open the AuctionManagerBean class in a source editor window.
2. Import the following class:

```
import auctionsystem.entity.BookItem;
```

3. Implement the addBookItem method. You can use the addItem method as a guide.
4. Build the AuctionApp project. Correct any errors.
5. Verify the AuctionApp project.
6. Deploy the AuctionApp application.

Task 3 – Testing the Creation of `Item` and `BookItem` Entries

In this task, you test the inheritance hierarchy by creating `Item` and `BookItem` entries. In this test, you use the `TestClient` class to create one or more entries in the `ITEM` and `BOOKITEM` tables in the database.

Complete the following steps:

1. Use a browser to load the following URL:

   `http://localhost:8080/AuctionApp/AuctionApp-app-client`

2. You should observe Java Webstart loading the `AuctionApp TestClient` GUI.

   You should observe the following message output on the `AuctionApp Log` window of the GUI:

```
Received hello on startup
```

3. Enter the following string in the AuctionApp TestClient GUI window.

```
addItem radio radio.jpg
```

   You should observe the following message output on your GUI window:

```
User input: addItem radio radio.jpg
Created auctionsystem.entity.Item[id=5]
```

4. Examine the `ITEM` table in the `Services` tab of the IDE.

   The `ITEM` table should have an entry for the book item you created. Note the value in the DISC column. It should be `Item`.

5. Create a book item by entering the following string in the AuctionApp TestClient GUI window.

```
addBookItem book mybook.jpg moose stanley
```

   You should observe the following message output on your GUI window:

```
User input: addBookItem book mybook.jpg moose stanley
Created auctionsystem.entity.BookItem[id=6]
```

6. Examine the `ITEM` and `BOOKITEM` table in the `Services` tab of the IDE.

   The `ITEM` table should have an entry for the book item you created. Note the value in the `DISC` column. It should be `BookItem`.

7. Optionally, you can create additional items and book items and observe the effect on the `ITEM` table.

Exercise 3: Completing Review Questions

In this exercise, you answer questions about the material covered in Module 6 of the Student Guide.

Complete the following questions:

1. What choices exist for selecting the entity classes superclass?
2. What object/relational mapping strategies are available to map entity classes that use inheritance?

Exercise Summary

Discussion – Take a few minutes to discuss what experiences, issues, or discoveries you had during the lab exercise.

- Experiences
- Interpretations
- Conclusions
- Applications

Lab 7

Using the Java Persistence Query Language (QL)

Objectives

Upon completion of this lab, you should be able to:

- Implement the `findAllOpenAuctions` use case
- Implement the `findAuctionsOfSeller` use case
- Implement the `showSeller` use case
- Complete the review questions

Exercise 1 – Implementing the findAllOpenAuctions Use Case

This exercise contains the following sections:

- Task 1 – Updating the Auction Entity Class
- Task 2 – Updating the AuctionManager Session Bean
- Task 3 – Testing the findAllOpenAuctions Use Case

Tool Reference – Tool references used in this exercise:

- Java EE Development: Enterprise Application Projects: Creating Enterprise Application Projects
- Java Development: Java Application Projects: Opening Projects
- Java EE Development: Enterprise Application Projects: Building Java EE Applications
- Java EE Development: Enterprise Application Projects: Verifying Java EE Applications
- Java EE Development: Enterprise Application Projects: Deploying Java EE Applications
- Java Development: Java Application Projects: Running Projects
- Server Resources: J2EE Application Servers: Examining Server Log Files
- Java Development: Modifying Project Libraries
- Java Development: Java Classes: Modifying Java Classes: Adding Fields
- Java Development: Java Classes: Opening Java Classes
- Java Development: Other Files: Opening Files
- Java EE Application Servers: Starting Application Servers
- Server Resources: Databases: Starting JavaDB
- Server Resources: Databases: Creating a Java DB Database
- Server Resources: Databases: Connecting to Databases

Task 1 – Updating the Auction Entity Class

In this task, you update the Auction entity class by annotating it with a NamedQuery annotation. Figure 7-1 shows the Auction entity in context.

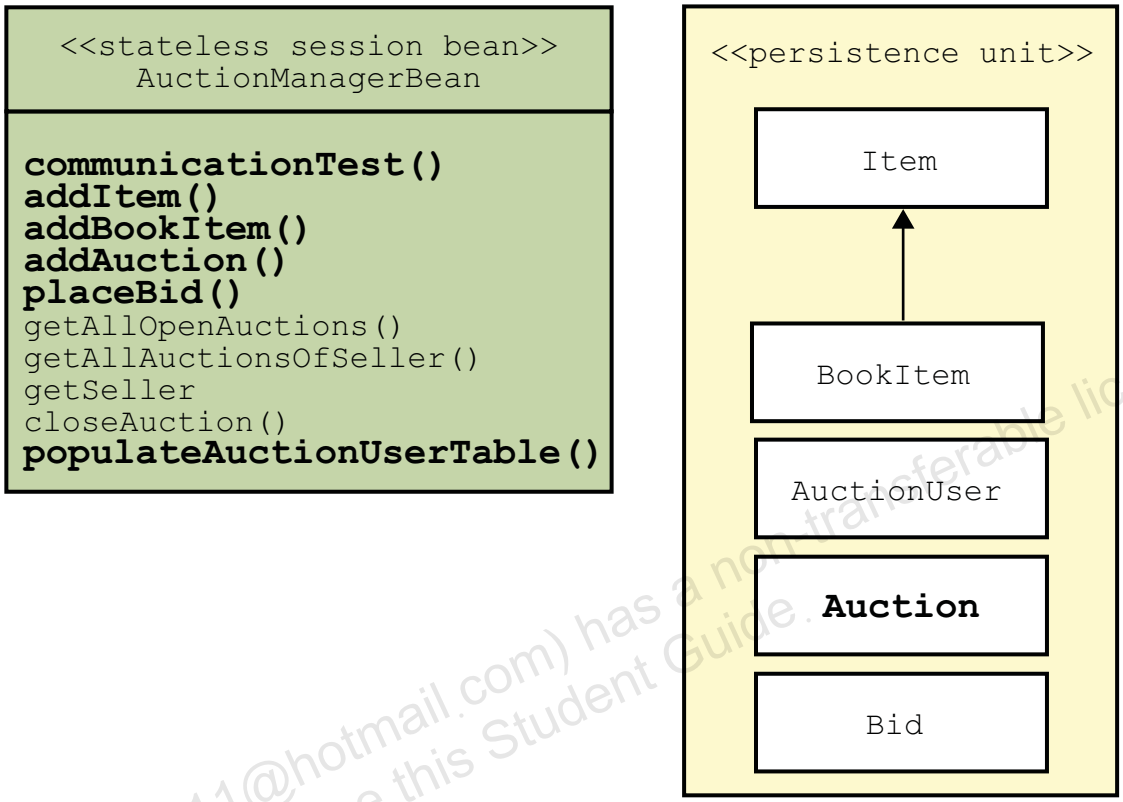

Figure 7-1 The Auction Entity

The NamedQuery annotation contains two elements, the name element and the query element. The name element is used to refer to the query when using the EntityManager methods that create query objects. The query element specifies the QL query to be executed. Table 7-1 contains the information you require to specify the NamedQuery annotation to service the findAllOpenAuctions use case.

Table 7-1 NamedQuery Annotation Information for findAllOpenAuctions Use Case

Element Name	Value
name	"FindAllOpenAuctions"
query	"SELECT OBJECT(a) FROM Auction AS a WHERE a.status= 'OPEN'"

Open the Auction entity class in a source editor window. Complete the following steps:

1. Import the following annotation.

```
import javax.persistence.NamedQuery;
```

2. Add the following lines of code between the Entity annotation and the class keyword:

```
@NamedQuery(name="FindAllOpenAuctions",
    query="SELECT OBJECT(a) FROM Auction AS a WHERE a.status='OPEN'")
```

3. Build the AuctionApp project. Correct any errors you encounter.
4. Verify the AuctionApp project.

Task 2 – Updating the AuctionManager Session Bean

In this task, you update the AuctionManager session bean by implementing the getAllOpenAuctions method. Figure 7-2 shows the getAllOpenAuctions method in context.

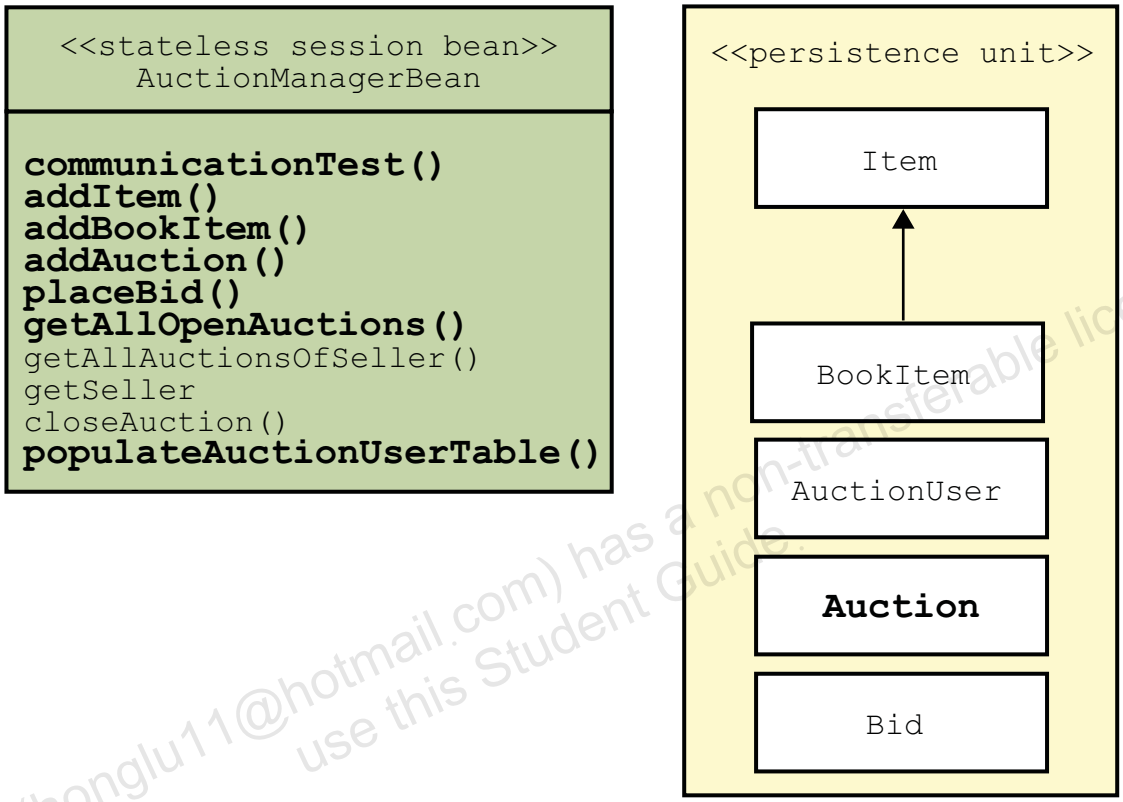

Figure 7-2 The getAllOpenAuctions Method of the AuctionManagerBean Class

Complete the following steps:

1. Open the AuctionManagerBean class in a source editor window.
2. Import the following class.

```
import javax.persistence.Query;
```

3. Use the following hints to implement the getAllOpenAuctions method.
   a. Declare a local variable named query of type Query.
   b. Use the createNamedQuery method of the entity manager to create a query instance of the FindAllOpenAuctions named query and assign it to the query object you declared in the previous step.

c. Execute the `getResultList` method on the query object and return the result after casting it to a `Collection` object.

4. Build the `AuctionApp` project. Correct any errors you encounter.
5. Verify the `AuctionApp` project.
6. Deploy the `AuctionApp` application.

Task 3 – Testing the findAllOpenAuctions Use Case

In this task, you test the `findAllOpenAuctions` use case.

Complete the following steps:

1. Use a browser to load the following URL:

   `http://localhost:8080/AuctionApp/AuctionApp-app-client`

2. You should observe Java Webstart loading the `AuctionApp TestClient` GUI.

   You should observe the following message output on the `AuctionApp Log` window of the GUI:

```
Received hello on startup
```

3. Enter the following string in the AuctionApp TestClient GUI window.

```
addAuction 5000 100 5 car car.jpg 3
```

   You should see the following output:

```
User input: addAuction 5000 100 5 car car.jpg 3
User #3 added an auction with the ID = 6 (Note: Actual ID might vary).
```

4. Create several more auctions.
5. Enter the following string in the AuctionApp TestClient GUI window.

```
showAllOpenAuctions
```

   You should see an output that shows all the open auctions in the auction application database.

Exercise 2 – Implementing the findAuctionsOfSeller Use Case (Optional)

This exercise contains the following sections:

- Task 1 – Updating the Auction Entity Class
- Task 2 – Updating the AuctionManager Session Bean
- Task 3 – Testing the findAuctionsOfSeller Use Case

You have the option to skip this exercise, because none of the following exercises depend on the completion of this exercise.

Tool Reference – Tool references used in this exercise:

- Java EE Development: Enterprise Application Projects: Creating Enterprise Application Projects
- Java Development: Java Application Projects: Opening Projects
- Java EE Development: Enterprise Application Projects: Building Java EE Applications
- Java EE Development: Enterprise Application Projects: Verifying Java EE Applications
- Java EE Development: Enterprise Application Projects: Deploying Java EE Applications
- Java Development: Java Application Projects: Running Projects
- Server Resources: J2EE Application Servers: Examining Server Log Files
- Java Development: Modifying Project Libraries
- Java Development: Java Classes: Modifying Java Classes: Adding Fields
- Java Development: Java Classes: Opening Java Classes
- Java Development: Other Files: Opening Files
- Java EE Application Servers: Starting Application Servers
- Server Resources: Databases: Starting JavaDB
- Server Resources: Databases: Creating a Java DB Database
- Server Resources: Databases: Connecting to Databases

Task 1 – Updating the Auction Entity Class

In this task, you update the Auction entity class by annotating it with a second NamedQuery annotation.

Complete the following steps:

1. Determine the query string to be used.

   The NamedQuery annotation contains two elements, the name element and the query element. The name element is used to refer to the query when using the EntityManager methods that create query objects. The query element specifies the QL query to be executed. Table 7-2 contains the name element of the NamedQuery annotation to service the findAuctionsOfSeller use case.

   Your first task is to determine the value of the query element string required to find auctions of a particular seller. The query string you require must use a parameter as the place holder for the seller ID.

Table 7-2 NamedQuery Annotation Information for findAuctionsOfSeller use Case

Element Name	Value
name	"FindAuctionsOfSeller"
query	

Open the Auction entity class in a source editor window. Complete the following steps.

1. Import the following annotation.

```
import javax.persistence.NamedQueries;
```

2. Declare a NamedQuery with the name findAuctionsOfSeller.

3. Wrap both NamedQuery annotations (findAllOpenAuctions and findAuctionsOfSeller) in a NamedQueries annotation.

   This step is required because multiple NamedQuery annotations cannot be added directly to an entity class. Instead, they must be wrapped in a NamedQueries annotation, as shown:

```
@NamedQueries({
  @NamedQuery(name="FindAllOpenAuctions",
    query=" SELECT OBJECT(a) FROM Auction AS a WHERE a.status= 'OPEN'"),
  @NamedQuery(name="FindAllAuctionsOfSeller",
    query=" SELECT OBJECT(a) FROM Auction AS a WHERE a.seller.id = ?1")
})
```

4. Build the AuctionApp project. Correct any errors you encounter.
5. Verify the AuctionApp project.

Task 2 – Updating the AuctionManager Session Bean

In this task, you update the AuctionManager session bean by implementing the getAllAuctionsOfSeller method. Figure 7-3 shows the getAllAuctionsOfSeller method in context.

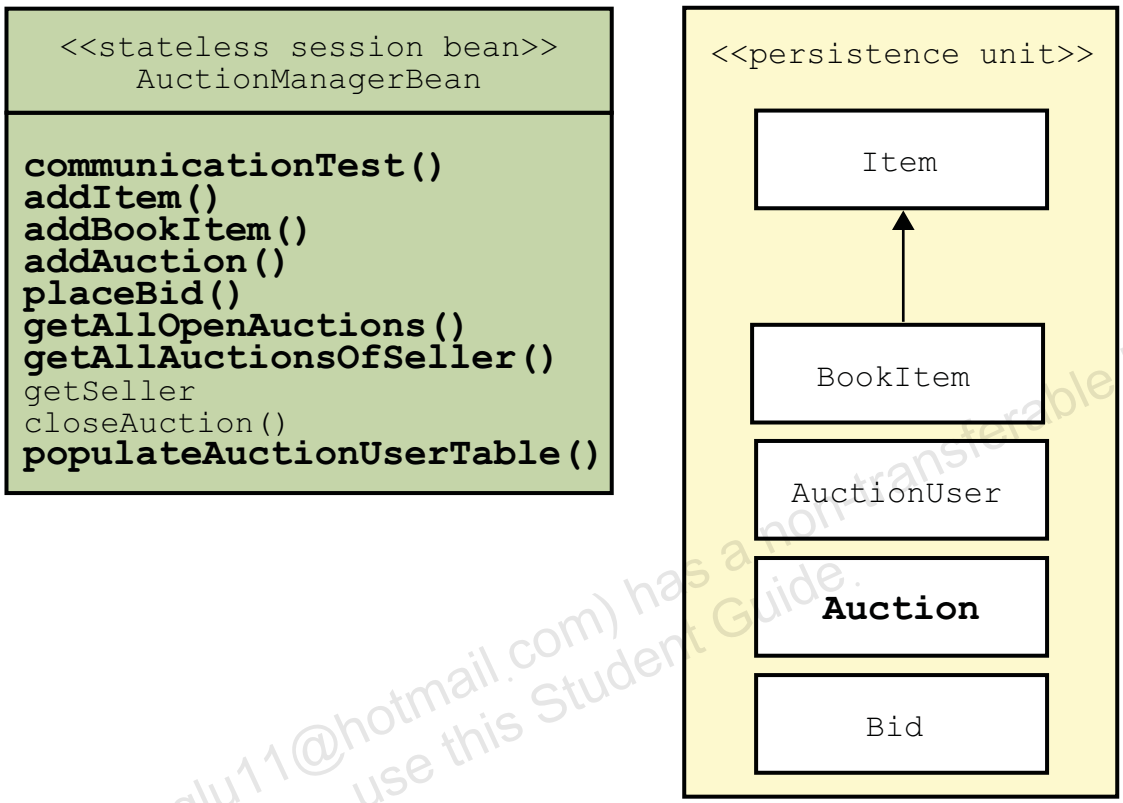

Figure 7-3 The getAllAuctionsOfSeller Method

Complete the following steps:

1. Open the AuctionManagerBean class in a source editor window.
2. Use the following hints to implement the getAllAuctionsOfSeller method.
   a. Declare a local variable named query of type Query.
   b. Use the createNamedQuery method of the entity manager to create a query instance of the FindAuctionsOfSeller named query and assign it to the query object you declared in the previous step.
   c. Use the setParameter method to pass the sellerID parameter value (of the getAllAuctionsOfSeller method) to the query object.
   d. Execute the getResultList method on the query object and return the result after casting it to a Collection object.

3. Build the AuctionApp project. Correct any errors you encounter.
4. Verify the AuctionApp project.
5. Deploy the AuctionApp application.

Task 3 – Testing the findAuctionsOfSeller Use Case

In this task, you test the findAuctionsOfSeller use case.

Complete the following steps:

1. Use a browser to load the following URL:

   ```
   http://localhost:8080/AuctionApp/AuctionApp-app-client
   ```

2. You should observe Java Webstart loading the AuctionApp TestClient GUI.

   You should observe the following message output on the AuctionApp Log window of the GUI:

```
Received hello on startup
```

3. Enter the following string in the AuctionApp TestClient GUI window.

```
addAuction 5000 100 5 car car.jpg 2
```

   You should see the following output:

```
User input: addAuction 5000 100 5 car car.jpg 2
User #2 added an auction with the ID = 6 (Note: Actual ID might vary).
```

4. Create several more auctions for auction users 3 and 4. Make a note of the number of auctions you create for each user.
5. Enter the following string in the AuctionApp TestClient GUI window:

```
showAuctionsOfSeller 2
```

   You should see an output that shows all auctions you created for auction user 2.

6. Repeat the previous steps for auction users 3 and 4.

Exercise 3 – Implementing the showSeller Use Case (Optional)

This exercise contains the following sections:

- Task 1 – Updating the `AuctionManager` Session Bean
- Task 2 – Testing the `getSeller` Use Case

You have the option to skip this exercise, because none of the following exercises depend on the completion of this exercise.

Tool Reference – Tool references used in this exercise:

- Java EE Development: Enterprise Application Projects: Creating Enterprise Application Projects
- Java Development: Java Application Projects: Opening Projects
- Java EE Development: Enterprise Application Projects: Building Java EE Applications
- Java EE Development: Enterprise Application Projects: Verifying Java EE Applications
- Java EE Development: Enterprise Application Projects: Deploying Java EE Applications
- Java Development: Java Application Projects: Running Projects
- Server Resources: J2EE Application Servers: Examining Server Log Files
- Java Development: Modifying Project Libraries
- Java Development: Java Classes: Modifying Java Classes: Adding Fields
- Java Development: Java Classes: Opening Java Classes
- Java Development: Other Files: Opening Files
- Java EE Application Servers: Starting Application Servers
- Server Resources: Databases: Starting JavaDB
- Server Resources: Databases: Creating a Java DB Database
- Server Resources: Databases: Connecting to Databases

Task 1 – Updating the AuctionManager Session Bean

In this task, you update the `AuctionManager` session bean by implementing the `getSeller` method. In the previous exercises, you declared a named query and created the query using the `createNamedQuery` method of the entity manager. In this task, you create a query object directly without first declaring a named query annotation. Figure 7-4 shows the `getSeller` method in context.

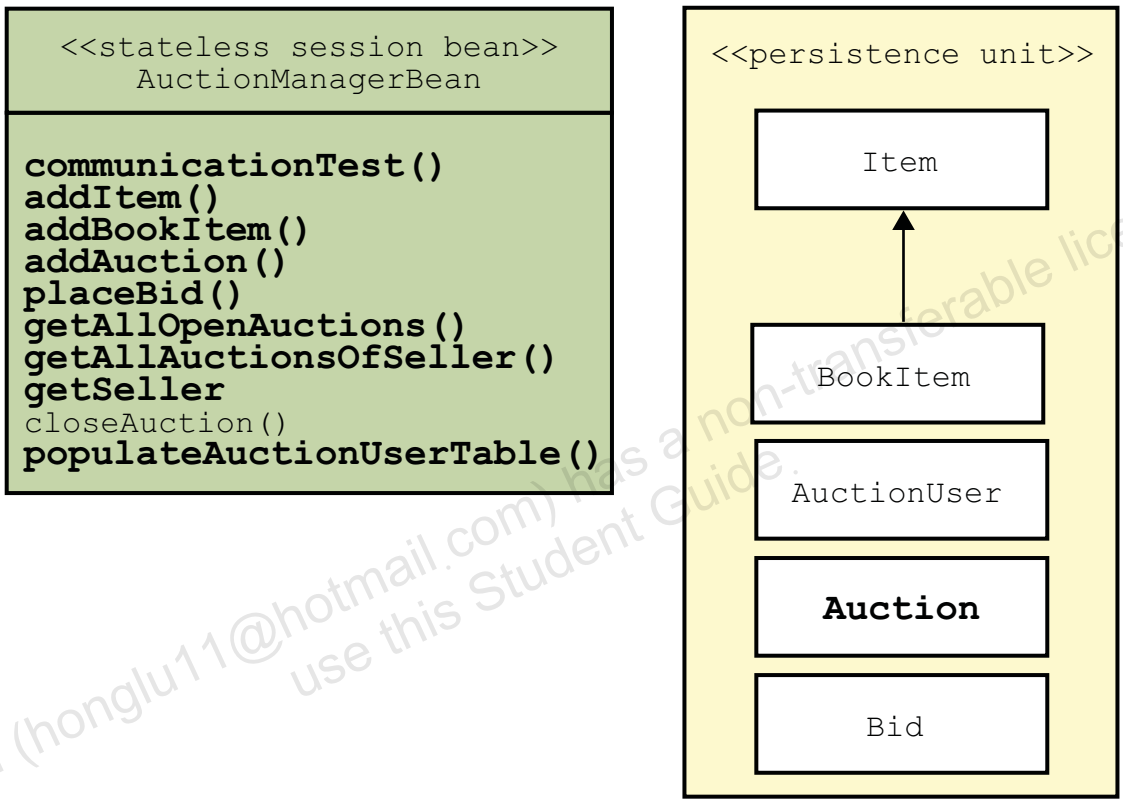

Figure 7-4 The `getSeller` Method

Complete the following steps:

1. Open the `AuctionManagerBean` class in a source editor window.

2. Type the code as shown in the code listing Code 7-1.

Code 7-1 Template Code for the getSeller Method

```
1   public Object getSeller(String displayName) {
2     String queryString =
3       "SELECT OBJECT(a) FROM AuctionUser AS a WHERE a.displayName = '"
4       + displayName +"'";
5     AuctionUser seller = null;
6     Query query = null;
7     try {
8       /*
9        Add code here
10      */
11      return seller;
12    } catch (Exception e){
13      throw new EJBException(e.getMessage());
14    }
15  }
```

3. Add code between the try catch block as follows:
   a. Use the `queryString` local variable to create a `Query` object using the `createQuery` method of the entity manager.
   b. Execute the `getSingleResult` method on the query object. and assign the result to seller after casting it to an `AuctionUser` object.
4. Build the `AuctionApp` project. Correct any errors you encounter.
5. Verify the `AuctionApp` project.
6. Deploy the `AuctionApp` application.

Task 2 – Testing the getSeller Use Case

In this task, you test the `findAllOpenAuctions` use case.

Complete the following steps:

1. Use a browser to load the following URL:

   `http://localhost:8080/AuctionApp/AuctionApp-app-client`

2. You should observe Java Webstart loading the `AuctionApp TestClient` GUI.

   You should observe the following message output on the `AuctionApp Log` window of the GUI:

```
Received hello on startup
```

3. Enter the following string in the AuctionApp TestClient GUI window. The following input assumes you have an auction user with the display name of `auctionman`.

```
showSeller auctionman
```

   You should see the following output:

```
Seller's auctionman personal data: auctionsystem.entity.AuctionUser[id=2]
```

Exercise 4: Completing Review Questions

In this exercise, you answer a question about the material covered in Module 7 of the Student Guide.

Complete the following question:

1. List the statement types supported by the Java persistence query language?

Exercise Summary

Discussion – Take a few minutes to discuss what experiences, issues, or discoveries you had during the lab exercise.

- Experiences
- Interpretations
- Conclusions
- Applications

Lab 8

Developing Java EE Applications Using Messaging

Objectives

Upon completion of this lab, you should be able to:

- Use the JMS API to create and send a message
- Complete the review questions

Exercise 1 – Sending Bid Status Messages Using JMS

In this exercise, you place a bid using JMS.

This exercise contains the following sections:

- Task1 – Preparing the Exercise Environment
- Task 2 – Adding JMS Message Sending to the `AuctionManager` Session Bean
- Task 3 – Copying New Client Classes
- Task 4 – Testing the Bid Status Message Sending Use Case

Preparation

This exercise assumes the application server and the Derby database server are installed and running.

Tool Reference – Tool references used in this exercise:

- Java EE Development: Enterprise Application Projects: Creating Enterprise Application Projects
- Java Development: Java Application Projects: Opening Projects
- Java EE Development: Enterprise Application Projects: Building Java EE Applications
- Java EE Development: Enterprise Application Projects: Verifying Java EE Applications
- Java EE Development: Enterprise Application Projects: Deploying Java EE Applications
- Java Development: Java Application Projects: Running Projects
- Server Resources: J2EE Application Servers: Examining Server Log Files
- Java Development: Modifying Project Libraries
- Java Development: Java Classes: Modifying Java Classes: Adding Fields
- Java Development: Java Classes: Opening Java Classes
- Java Development: Other Files: Opening Files
- Java EE Application Servers: Starting Application Servers

- Server Resources: Databases: Starting JavaDB
- Server Resources: Databases: Creating a Java DB Database
- Server Resources: Databases: Connecting to Databases
- Server Resources: Messaging: JMS Resource
- Server Resources: Java EE Application Servers: Administrating JMS: Creating Physical Destinations
- Server Resources: Configuring Java EE Resources: Configuring JMS Connection Factories
- Server Resources: Configuring Java EE Resources: Configuring JMS Destination Resources

Task1 – Preparing the Exercise Environment

This task requires you to use the admin console of the Java EE application server to create the following.

1. A physical topic destination named `bidsOutTopic`.
2. A JMS topic connection factory resource.
3. A JMS destination resource.

Figure 8-1 shows the JMS topic connection factory, the JMS destination resource and the physical topic destination.

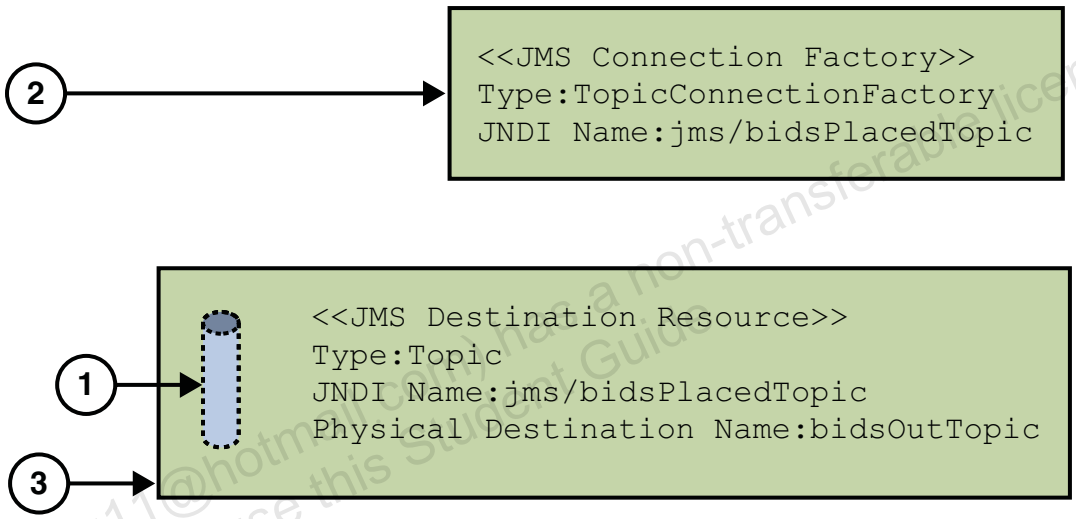

Figure 8-1 The JMS Topic Resources

Complete the following steps:

1. Create a new JMS Physical Destination with the following characteristics:

   a. Bring up the application server's admin console by entering the following URL in a browser:

   `http://localhost:4848`

   Log in using the following information:

   User name: **`admin`**

   Password: **`adminadmin`**

   b. Using the application server's admin console expand the nodes to view the Physical Destination management page:

`Configuration > Java Message Service > Physical Destinations`

c. Press the New button to obtain a New Physical Destination dialog box.

d. Enter the following information in the ensuing New Physical Destination dialog box.

Physical Destination Name: **`bidsOutTopic`**

Type: **`topic`**

2. Create a new JMS `ConnectionFactory` with the following characteristics:

a. Use the application server's admin console and expand the following nodes to view the JMS Connection Factories page:

```
Resources > JMS Resources > Connection Factories
```

b. Press the New button to obtain a New JMS Connection Factory dialog box.

c. Enter the following information in the ensuing New JMS Connection Factory dialog box.

JNDI Name: **`jms/TopicConnectionFactory`**

Type: **`javax.jms.TopicConnectionFactory`**

Resource: **`Enabled`**

Transaction Support: **`XATransaction`**

Properties:

User name: **`guest`**

Password: **`guest`**

3. Create a new JMS Destination resource with the following characteristics:

a. Use the application server's admin console and expand the following nodes to view the `JMS Destination Resources` page:

```
Resources > JMS Resources > Destination Resources
```

b. Press the New button to obtain a New JMS Destination Resource dialog box.

c. Enter the following information in the ensuing Edit JMS Connection Factory dialog box.

JNDI Name: **`jms/bidsPlacedTopic`**

Type: **`javax.jms.Topic`**

Enabled: **True**

Physical Destination Name: **`bidsOutTopic`**

d. Press the OK button to save the information.

Task 2 – Adding JMS Message Sending to the AuctionManager Session Bean

In this task, you update the AuctionManager session bean by implementing the sendBidStatusUpdateMessage method. The sendBidStatusUpdateMessage method sends bid information wrapped in a BidsStatusMessage object to the bidsOutTopic. Figure 8-2 illustrates the transfer of the message from the server to the client. The TestClient subscribes to bidsOutTopic and when notified, the TestClient fetches BidStatusMessage objects from bidsOutTopic.

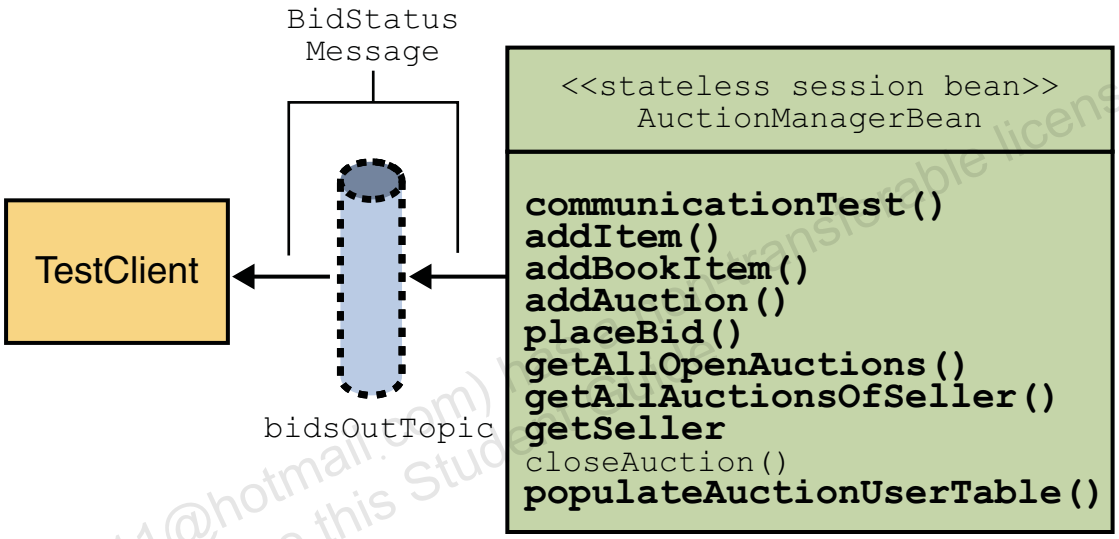

Figure 8-2 Message Transfer Between the AuctionManagerBean Class and the TestClient Class

Open the AuctionManagerBean class in a source editor window. Complete the following steps.

1. Import the following classes.

```
import javax.jms.Connection;
import javax.jms.TopicConnectionFactory;
import javax.jms.Topic;
import javax.jms.Session;
import javax.jms.MessageProducer;
import javax.jms.ObjectMessage;
import javax.jms.JMSException;
import auctionsystem.dto.BidStatusMessage;
import javax.annotation.Resource;
```

2. Add the following resource injection/attribute declaration statements to the attributes section.

```
@Resource(mappedName = "jms/bidsPlacedTopic")
private Topic bidsPlacedTopic;
@Resource(mappedName = "jms/TopicConnectionFactory")
private TopicConnectionFactory connectionFactory;
```

3. Type the code for the `sendBidStatusUpdateMessage`, as shown in the code listing Code 8-1.

Code 8-1 Template Code for the `sendBidStatusUpdateMessage` Method

```
1   private void sendBidStatusUpdateMessage(String auctionStatus,
2     Integer auctionID,Integer bidderID, double amount) {
3     TopicConnection connection = null;
4     Session session = null;
5     MesssgeProducer producer = null;
6     ObjectMessage message = null;
7     int property = auctionID.intValue();
8     BidStatusMessage bidMessage =
9       new BidStatusMessage(auctionStatus,bidderID,amount);
10    try {
11      connection = // Todo create connection
12      session = // Todo use connection to createSession(true, 0)
13      producer =//Todo Use session & bidsPlacedTopic to create producer
14      message = // Todo create message using session and bidMessage
15      message.setIntProperty("auctionID", property);
16      // Todo use producer to send message
17    } catch (JMSException je) {
18      throw new EJBException(je);
19    } finally {
20      if (connection != null) {
21        try {
22          connection.close();
23        } catch (JMSException je) {
24          throw new EJBException(je);
25        } finally {
26          connection = null;
27        }
28      } // end if
29    } // end finally
30  } // end method bidStatusUpdateMessage
```

4. Use the hints in the comments beginning with `Todo` to complete the respective lines of code.

5. Invoke the `sendBidStatusUpdateMessage` from the end of the `placeBid` method.

6. Build the `AuctionApp` project. Correct any errors you encounter.

7. Verify the `AuctionApp` project.

Task 3 – Copying New Client Classes

Complete the following steps:

1. Use the Files tab to locate and delete the `TestClient.java` file used previously. The path to this file is:

`AuctionApp-app-client/src/java/auctionapp`

2. Open the Favorites tab and locate the following directory:

`exercises/mod08_jms/exercise1`

   Copy the following file:

   - `TestClient.java`

3. Use the Files tab to paste the `TestClient.java` file in the following directory:

`AuctionApp-app-client/src/java/auctionapp`

4. Build the `AuctionApp` project. Correct any errors you encounter.
5. Verify the `AuctionApp` project.
6. Deploy the `AuctionApp` application.

Task 4 – Testing the Bid Status Message Sending Use Case

In this task, you test the `findAllOpenAuctions` use case.

Complete the following steps:

1. Use a browser to load the following URL:

   `http://localhost:8080/AuctionApp/AuctionApp-app-client`

2. You should observe Java Webstart loading the `AuctionApp TestClient` GUI.

   You should observe the following message output on the `AuctionApp Log` window of the GUI:

```
Received hello on startup
```

3. Enter the following string in the AuctionApp TestClient GUI window.

```
addAuction 5000 100 5 car car.jpg 3
```

   You should see the following output:

```
User input: addAuction 5000 100 5 car car.jpg 3
User #3 added an auction with the ID = 6 (Note: Actual ID might vary).
```

4. Enter the following string in the AuctionApp TestClient GUI window.

```
receiveBidStatusMessage 6
```

   You should see the output similar to the following:

   `registered to participate in auction # 6`

Note – The number 6 should be the auction ID returned in Step 3.

5. Enter the following string in the AuctionApp TestClient GUI window.

```
placeBid 6 2 6000
```

   The output generated by this command is:

```
User #2 has placed a bid
```

Note – Due to the multithreaded design of the `TestClient` applications, in some cases the output generated by Step 5 might be preceded by the output from Step 6.

6. Verify the reception of the corresponding bid status message by examining the AuctionApp TestClient GUI window. You should see an output similar to the following.

```
--> Heard there was a message in TestClient!!
   You are registered for bid status messages on
auction #6
    a bid message was received:

  auction status = OPEN
  BidderID = 2
  bidAmount = 6000.0
```

7. Repeat the previous two steps.

Exercise 2: Completing Review Questions

In this exercise, you answer questions about the material covered in Module 8 of the Student Guide.

Complete the following questions:

1. List the two types of messaging destinations?
2. List the three parts of a JMS message?

Exercise Summary

Discussion – Take a few minutes to discuss what experiences, issues, or discoveries you had during the lab exercise.

- Experiences
- Interpretations
- Conclusions
- Applications

Lab 9

Developing Message-Driven Beans

Objectives

Upon completion of this lab you should be able to:

- Place a bid using a message-driven bean
- Complete the review questions

Exercise 1 – Placing a Bid Using a Message-Driven Bean

In this exercise, you place a bid using a message-driven bean.

This exercise contains the following sections:

- Task 1 – Updating the `AuctionManagerLocal` Session Bean Interface
- Task 2 – Creating the `PlaceBid` Message-Driven Bean
- Task 3 – Copying New Client Classes
- Task 4 – Testing the Bid Status Message Sending Use Case

Preparation

This exercise assumes the application server and the JavaDB database server are installed and running.

Tool Reference – Tool references used in this exercise:

- Java EE Development: Enterprise Application Projects: Creating Enterprise Application Projects
- Java Development: Java Application Projects: Opening Projects
- Java EE Development: Enterprise Application Projects: Building Java EE Applications
- Java EE Development: Enterprise Application Projects: Verifying Java EE Applications
- Java EE Development: Enterprise Application Projects: Deploying Java EE Applications
- Java Development: Java Application Projects: Running Projects
- Server Resources: J2EE Application Servers: Examining Server Log Files
- Java Development: Modifying Project Libraries
- Java Development: Java Classes: Modifying Java Classes: Adding Fields
- Java Development: Java Classes: Opening Java Classes
- Java Development: Other Files: Opening Files
- Java EE Application Servers: Starting Application Servers
- Server Resources: Databases: Starting JavaDB

- Server Resources: Databases: Creating a Java DB Database
- Server Resources: Databases: Connecting to Databases
- Server Resources: Messaging: JMS Resource
- Server Resources: Java EE Application Servers: Administrating JMS: Creating Physical Destinations
- Server Resources: Configuring Java EE Resources: Configuring JMS Connection Factories
- Server Resources: Configuring Java EE Resources: Configuring JMS Destination Resources

Task 1 – Updating the AuctionManagerLocal Session Bean Interface

In this task, you update the AuctionManagerLocal interface by adding the placeBid method signature. Figure 9-1 shows the addition of the placeBid method to the AuctionManagerLocal interface.

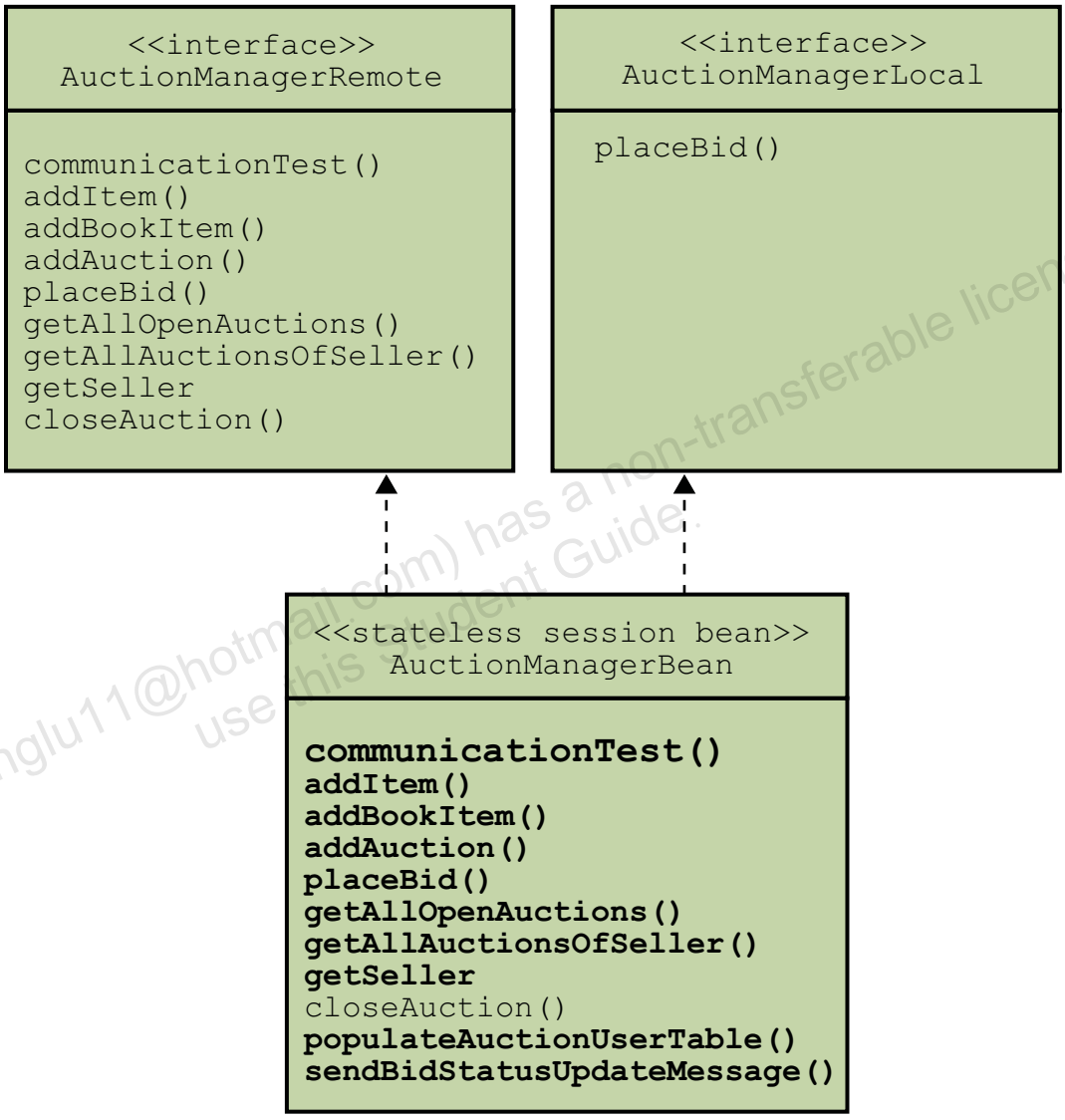

Figure 9-1 The AuctionManagerLocal Interface Showing the placeBid Method

Open the `AuctionManagerLocal` interface in a source editor window. Complete the following steps:

1. Import the following class.

```
import auctionsystem.exception.AuctionException;
```

2. Add the `placeBid` method signature.

```
public void placeBid(Integer auctionID, Integer bidderID,
  double bidAmount) throws AuctionException;
```

3. Build the `AuctionApp` project. Correct any errors you encounter.
4. Verify the `AuctionApp` project.

Task 2 – Creating the PlaceBid Message-Driven Bean

In this task, you create the `PlaceBidMD` message-driven bean. Figure 9-2 illustrates the interaction between the `autoBid` method in the `TestClient` class, the `placeBidMD` queue, the `placeBidMD` message-driven bean, and the `AuctionManagerBean` session bean. The `autoBid` method creates a bid and sends it wrapped in a `PlaceBidMessage` object to the `placeBidMD` queue. The EJB container forwards the `PlaceBidMessage` object to the `onMessage` method of the `PlaceBidMD` message-driven bean, The `onMessage` method places the bid using the `AuctionManagerLocal` interface's `placeBid` method.

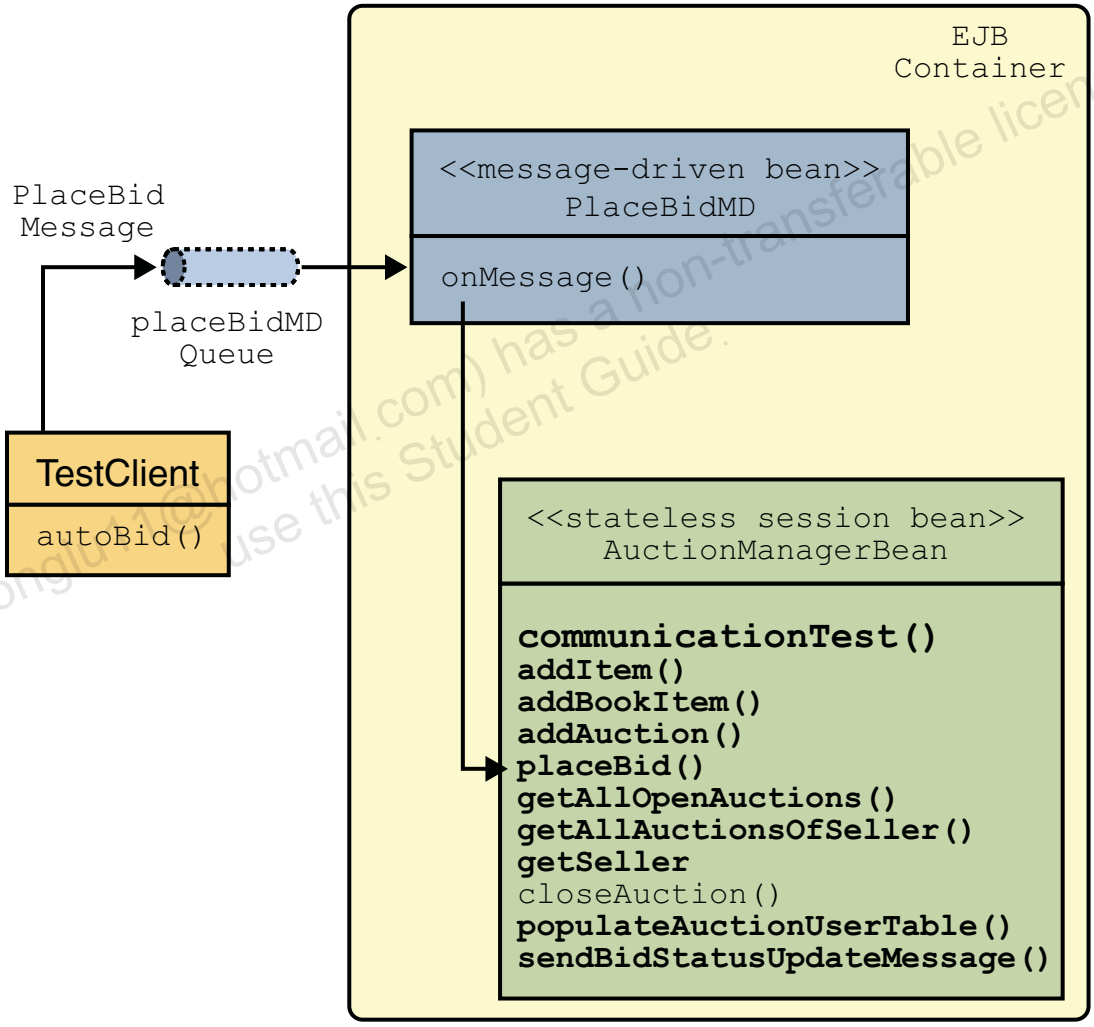

Figure 9-2 Message Transfer Between the `TestClient` Class and the `AuctionManagerBean` Session Bean

Complete the following steps:

1. Use the Projects window of the IDE to obtain a New Message-Driven Bean dialog box.
2. Create a message-driven bean with the following characteristics:

   Name: **`PlaceBidMD`**

   Project: **`AuctionApp-ejb`**

   Location: **`Source Packages`**

   Package: **`auctionsystem.ejb`**

   Project Destinations: Use the `Add` button and enter the following values

   Destination Name: **`PlaceBidMD`**

   Destination Type: **`Queue`**

Note – The deployment of the `PlaceBidMD` message-driven bean creates the following resources on the application server:
A physical destination with the name: `PlaceBidMD`
A JMS connection factory with the JNDI name: `jms/PlaceBidMDFactory`
A JMS queue destination resource with the JNDI name: `jms/PlaceBidMD`

3. Import the following classes:

```
import javax.jms.ObjectMessage;
import javax.jms.JMSException;
import javax.ejb.EJBException;
import javax.ejb.EJB;
import auctionsystem.exception.AuctionException;
import auctionsystem.dto.PlaceBidMessage;
```

4. Use injection (EJB annotation) to obtain a reference to an instance of the `AuctionManagerLocal` interface.

```
@EJB AuctionManagerLocal auctionManager;
```

5. Implement the `onMessage` method by completing the following steps:

   a. Declare the following local variables.

```
ObjectMessage objectMessage = null;
PlaceBidMessage placeBidMessage = null;
Integer auctionID = null;
Integer bidderID = null;
double amount = 0.0;
```

b. Assign (by casting to `ObjectMessage` type) the content of the `onMessage` method's argument message to the local variable `objectMessage`.

   The assignment statement should be wrapped in a try catch block to catch the `ClassCastException`.

c. Extract, into the local variable `placeBidMessage`, an object of type `PlaceBidMessage` from the `objectMessage`.

   The assignment statement should be wrapped in a try catch block to catch the `JMSException`.

d. Assign values to the local variables `auctionID`, `bidderID`, and `amount` by extracting the corresponding data from the `placeBidMessage` object.

e. Invoke the `placeBid` method of the `AuctionManagerLocal` instance.

   The method invocation should be wrapped in a try catch block to catch the `AuctionException`.

6. Build the `AuctionApp` project. Correct any errors you encounter.
7. Verify the `AuctionApp` project.

Task 3 – Copying New Client Classes

In this task, you replace the `TestClient` class with a newer version.

Complete the following steps:.

1. Use the Files tab to locate and delete the `TestClient.java` file used previously. The path to this file is:

`AuctionApp-app-client/src/java/auctionapp`

2. Open the Favorites tab and locate the following directory:

`exercises/mod09_mdb/exercise1`

   Copy the file: `TestClient.java`

3. Use the Files tab to paste the `TestClient.java` file in the following directory:

`AuctionApp-app-client/src/java/auctionapp`

4. Build the `AuctionApp` project. Correct any errors you encounter.
5. Verify the `AuctionApp` project.
6. Deploy the `AuctionApp` application.

Task 4 – Testing the Bid Status Message Sending Use Case

In this task, you test the `findAllOpenAuctions` method.

Complete the following steps:

1. Use a browser to load the following URL:

   `http://localhost:8080/AuctionApp/AuctionApp-app-client`

2. You should observe Java Webstart loading the `AuctionApp TestClient` GUI.

   You should observe the following message output on the `AuctionApp Log` window of the GUI:

```
Received hello on startup
```

3. Enter the following string in the AuctionApp TestClient GUI window.

```
addAuction 5000 100 5 car car.jpg 3
```

   You should see the following output:

```
User input: addAuction 5000 100 5 car car.jpg 3
User #3 added an auction with the ID = 6 (Note: Actual ID might vary).
```

4. Enter the following string in the AuctionApp TestClient GUI window.

```
placeBid 6 2 5500
```

   You should see the following output:

```
User input: placeBid 6 2 5500
 User #2 has placed a bid
```

5. Use the `Services` tab of the IDE to confirm that the `ITEM` table shows a bid value of 5500 placed by auction user 1.

6. Enter the following string in the AuctionApp TestClient GUI window.

```
autoBid 6 4 5750 7000 250
```

You should see the output similar to the following on the GUI window:

```
--> Attempting to start an auto-bid session
   Auction: 6 Bidder : 4 Max Bid: 7000.0   Bid Imt:
250.0
 Auto-Bid created Successfully.
   --> Heard there was a message in TestClient!!
   You are registered to autobid on auction #5
    a bid message was received:

  auction status = OPEN
  BidderID = 4
  bidAmount = 5750.0
     New Bid: 5750.0   Bidder: 4 Status: OPEN
     Your Auto-Bid Definition
   Bidder : 4 Max Bid: 7000.0   Bid Imt: 250.0
```

Note – The `autoBid` feature, as invoked in this step, places an automatic bid on auction 6, on behalf of auction user 4, beginning with $5750 in increments of $250 up to a maximum $7000.

7. Use the `Services` tab of the IDE to confirm that the `BID` table shows a new bid value of 5750 placed by the autobidder on behalf of auction user 4.
8. Enter the following string in the AuctionApp TestClient GUI window.

`placeBid 6 2 6000`

You should see the following output:

```
User #2 has placed a bid
```

9. Verify the reception of the corresponding bid status message by examining the GUI window. You should see an output similar to the following.

```
 --> Heard there was a message in TestClient!!
   You are registered to autobid on auction #6
    a bid message was received:

  auction status = OPEN
  BidderID = 4
  bidAmount = 6250.0
     New Bid: 6250.0   Bidder: 4 Status: OPEN
     Your Auto-Bid Definition
   Bidder : 4 Max Bid: 7000.0   Bid Imt: 250.0
```

10. Use the `Services` tab of the IDE to confirm that the `BID` table shows two new bids. The first of these would be a bid value of 6000 placed by auction user 1. This is followed by a bid of 6250 placed by the auto bidder on behalf of auction user 4.
11. You can optionally place additional bids on behalf of auction user 1 and observe the reaction of the auto bidder.
12. Examine the code of the `autoBid` method (and the `onMessage` method) of the TestClient class and note that it sends bids using JMS messaging to the `PlaceBidMD` queue.

Exercise 2: Completing Review Questions

In this exercise, you answer questions about the material covered in Module 9 of the Student Guide.

Complete the following questions:

1. True or false: A message-driven bean is a synchronous message consumer?
2. Name the method of the JMS message-driven bean that is called by the server when a message arrives?

Exercise Summary

Discussion – Take a few minutes to discuss what experiences, issues, or discoveries you had during the lab exercise.

- Experiences
- Interpretations
- Conclusions
- Applications

Lab 10

Implementing Interceptors

Objectives

Upon completion of this lab, you should be able to:

- Complete the review questions

Exercise 1: Completing Review Questions

In this exercise, you answer questions about the material covered in Module 10 of the Student Guide.

Complete the following questions:

1. Can a business interceptor method be associated with a specific business method?
2. When multiple interceptors are defined, what is the order of their invocation?

Implementing Transactions

Objectives

Upon completion of this lab, you should be able to:

- Complete the review questions

Note – The transaction exercises are combined with the exceptions exercises. See Lab 12, "Handling Exceptions."

Exercise 1: Completing Review Questions

In this exercise, you answer questions about the material covered in Module 11 of the Student Guide.

Complete the following questions:

1. What is the default transaction demarcation attribute?
2. Which method of the `EJBContext` object would you use to request a rollback of a transaction?

Lab 12

Handling Exceptions

Objectives

Upon completion of this lab, you should be able to:

- Handle exceptions created by transaction rollbacks
- Complete the review questions

Exercise 1 – Managing the Transaction Demarcation of the addAuction Use Case

In this exercise, you identify the methods that participate in the addAuction use case and specify their transaction demarcation properties.

This exercise contains the following sections:

- Task 1 – Examining Transaction Demarcation: Using Default Values
- Task 2 – Examining Transaction Demarcation: Using RequiresNew
- Task 3 – Examining Transaction Demarcation: Using Container Services
- Task 4 – Finalizing the Transaction Demarcation for the addAuction Use Case

Tool Reference – Tool references used in this exercise:

- Java EE Development: Enterprise Application Projects: Creating Enterprise Application Projects
- Java Development: Java Application Projects: Opening Projects
- Java EE Development: Enterprise Application Projects: Building Java EE Applications
- Java EE Development: Enterprise Application Projects: Verifying Java EE Applications
- Java EE Development: Enterprise Application Projects: Deploying Java EE Applications
- Java Development: Java Application Projects: Running Projects
- Server Resources: J2EE Application Servers: Examining Server Log Files
- Java Development: Modifying Project Libraries
- Java Development: Java Classes: Modifying Java Classes: Adding Fields
- Java Development: Java Classes: Opening Java Classes
- Java Development: Other Files: Opening Files
- Java EE Application Servers: Starting Application Servers
- Server Resources: Databases: Starting JavaDB
- Server Resources: Databases: Connecting to Databases

Task 1 – Examining Transaction Demarcation: Using Default Values

In this task, you test the impact of using default transaction demarcation values for the methods of the `addAuction` use case.

Complete the following steps:

1. Examine the source code for the `addAuction` method in the `AuctionManager` session bean class.

   Your objective is to identify all methods of the `AuctionManager` session bean class that participate in the creation of an auction entry in the database.

2. Fill the values for default transaction attribute column of Table 12-1.

Table 12-1 Default Transaction Attributes for Methods in the `addAuction` Use Case

Method Name	Default Transaction Attribute
`addAuction`	
`addItem`	

3. Deploy the `AuctionApp` application.
4. Use a browser to load the following URL:

   `http://localhost:8080/AuctionApp/AuctionApp-app-client`

5. You should observe Java Webstart loading the `AuctionApp TestClient` GUI.

   You should observe the following message output on the `AuctionApp Log` window of the GUI.:

```
Received hello on startup
```

6. Enter the following string in the AuctionApp TestClient GUI window.

```
addAuction 5000 100 5 car car.jpg 2
```

   You should see the following output:

```
User input: addAuction 5000 100 5 car car.jpg 2
User #2 added an auction with the ID = 6 (Note: Actual ID might vary).
```

7. Use the Services tab of the IDE to examine the AUCTION and ITEM tables.

   Verify that an entry has been created in each table.

8. Enter the following string in the AuctionApp TestClient GUI window.

```
addAuction 300 10 5 bike bike.jpg 333
```

   You should see the following output:

```
User input: addAuction 300 10 5 bike bike.jpg 333
nested exception is: java.rmi.ServerException: RemoteException occurred
in server thread; nested exception is:
    java.rmi.RemoteException: Unknown seller ID 333:
```

9. Use the Services tab of the IDE to examine the AUCTION and ITEM tables.

   Verify that no new entry has been created in either table.

10. Review the code for the addAuction method of the AuctionManager session bean. Consider the following:

    - The addItem method was called from the addAuction and it returned without an exception back to the addAuction method.
    - The successful completion of the addItem method should have created an entry for the bike item in the ITEM table.
    - Identify the line of code that threw the exception

Discussion – Consider the following:

- Why is there no entry for the bike item in the ITEM table.?

Task 2 – Examining Transaction Demarcation: Using RequiresNew

In this task, you test impact of using RequiresNew transaction demarcation values for the methods of the addAuction use case.

Complete the following steps:

1. Examine the source code for the addAuction method in the AuctionManager session bean class.

   Set the transaction attribute of the addAuction method and the addItem method to RequiresNew, as shown in Table 12-2.

Table 12-2 Alternative Transaction Attributes for Methods in the addauction Use Case

Method Name	Transaction Attribute
addAuction	RequiresNew
addItem	RequiresNew

   This task requires you to prepend each of the methods with the following line of code.

```
@TransactionAttribute(TransactionAttributeType.REQUIRES_NEW)
```

   Remember to also import the annotations.

2. Build the AuctionApp project. Correct any errors you encounter.
3. Verify the AuctionApp project.
4. Deploy the AuctionApp application.
5. Use a browser to load the following URL:

   http://localhost:8080/AuctionApp/AuctionApp-app-client

6. You should observe Java Webstart loading the AuctionApp TestClient GUI.

   You should observe the following message output on the AuctionApp Log window of the GUI.:

```
Received hello on startup
```

7. Enter the following string in the AuctionApp TestClient GUI window.

```
addAuction 5000 100 5 car car.jpg 2
```

You should see the following output:

```
User input: addAuction 5000 100 5 car car.jpg 2
User #2 added an auction with the ID = 6 (Note: Actual ID might vary).
```

8. Use the Services tab of the IDE to examine the AUCTION and ITEM tables.

   Verify that an entry has been created in each table.

9. Enter the following string in the AuctionApp TestClient GUI window.

```
addAuction 300 10 5 bike bike.jpg 333
```

   You should see the following output:

```
User input: addAuction 300 10 5 bike bike.jpg 333
nested exception is: java.rmi.ServerException: RemoteException occurred
in server thread; nested exception is:
    java.rmi.RemoteException: Unknown seller ID 333:
```

10. Use the Services tab of the IDE to examine the AUCTION and ITEM tables.

    Verify that no new entry has been created in either table.

Discussion – Consider the following:

- Why is there no entry for the bike item in the ITEM table?

  The transaction attributes for the addItem and addAuction methods specify that each method should be executed in a separate transaction. If this were the case, you should have observed a new entry for the bike item in the ITEM table, but no entry for a bike auction in the AUCTION table.

Task 3 – Examining Transaction Demarcation: Using Container Services

In this exercise, you use the local interface of the AuctionManger to invoke the addItem method from the addAuction method. Invoking a method through an EJB interface enables the container to intercept and manage the method invocation.

Complete the following steps:

1. Add the following method signature to the AuctionManagerLocal interface.

```
Object addItem(String itemDesc, String itemImage);
```

2. Add the following injected instance variable declaration to the AuctionManagerBean class.

```
@Resource private SessionContext sessionContext;
```

3. Locate the addAuction method of the AuctionManagerBean class. Replace the direct invocation of the addItem method with the following lines of code:

```
AuctionManagerLocal auctionMgrLocal =
  sessionContext.getBusinessObject(AuctionManagerLocal.class);
item = (Item)auctionMgrLocal.addItem(itemDesc, itemImage);
```

4. Build the AuctionApp project. Correct any errors you encounter.
5. Verify the AuctionApp project.
6. Deploy the AuctionApp application.
7. Use a browser to load the following URL:

   `http://localhost:8080/AuctionApp/AuctionApp-app-client`

8. You should observe Java Webstart loading the AuctionApp TestClient GUI.

   You should observe the following message output on the AuctionApp Log window of the GUI:

```
Received hello on startup
```

9. Enter the following string in the AuctionApp TestClient GUI window.

```
addAuction 5000 100 5 car car.jpg 2
```

You should see the following output:

```
User input: addAuction 5000 100 5 car car.jpg 2
User #2 added an auction with the ID = 6 (Note: Actual ID might vary).
```

10. Use the Services tab of the IDE to examine the AUCTION and ITEM tables.

    Verify an entry has been created in each table.

11. Enter the following string in the AuctionApp TestClient GUI window.

```
addAuction 300 10 5 bike bike.jpg 333
```

    You should see the following output:

```
User input: addAuction 300 10 5 bike bike.jpg 333
nested exception is: java.rmi.ServerException: RemoteException occurred
in server thread; nested exception is:
    java.rmi.RemoteException: Unknown seller ID 333:
```

12. Use the Services tab of the IDE to examine the AUCTION and ITEM tables.

    Verify that the AUCTION table has one entry for the car auction.

    Verify that the ITEM table has two entries, one for the car auction and a second for the bike auction.

Discussion – Consider the following:

- Why does the addAuction use case have to be performed in one transaction?

  Discuss the four transactional principles: atomic, consistent, isolated, and durable (ACID) for the addAuction use case.

- What are the implications of the addAuction use case implementation's exception handling on the transaction state?

Task 4 – Finalizing the Transaction Demarcation for the addAuction Use Case

In this exercise, you set the transaction demarcation for the `addItem` and `addAuction` methods so that both methods run in the same transaction.

Complete the following steps:

1. Select the transaction demarcation attribute for the `addItem` and `addAuction` methods so that both methods run in the same transaction. Your options for the transaction demarcation attribute are:

   ```
   TransactionAttributeType.REQUIRED
   TransactionAttributeType.REQUIRES_NEW
   TransactionAttributeType.Supports
   TransactionAttributeType.Mandatory
   TransactionAttributeType.NotSupported
   TransactionAttributeType.Never
   ```

2. Update the `addItem` and `addAuction` methods.
3. Build the `AuctionApp` project. Correct any errors you encounter.
4. Verify the `AuctionApp` project.
5. Deploy the `AuctionApp` application.
6. Test the `AuctionApp` application.

Exercise 2: Completing Review Questions

In this exercise, you answer a question about the material covered in Module 12 of the Student Guide.

Complete the following question:

1. Which annotation should you use when declaring an application exception?

Exercise Summary

Discussion – Take a few minutes to discuss what experiences, issues, or discoveries you had during the lab exercise.

- Experiences
- Interpretations
- Conclusions
- Applications

Lab 13

Using Timer Services

Objectives

Upon completion of this lab, you should be able to:

- Close an auction using a time out
- Complete the review questions

Exercise 1 – Use Case Overview: Closing an Auction Using a Time Out

In this exercise, you work on the `addAuction` use case and the `closeAuction` use case. You provide the auction system with new functionality that causes an auction to close automatically after its open period has expired.

This exercise focuses on adding timer functionality to an EJB component.

This exercise contains the following sections:

- Task 1 – Implementing the Close Auction Functionality
- Task 2 – Testing the Close Auction Functionality
- Task 3 – Implementing the `CreateTimer` Object Functionality
- Task 4 – Implementing the `Timeout` Event Handler Method
- Task 5 – Testing the Close Auction Timer Functionality

Tool Reference – Tool references used in this exercise:

- Java EE Development: Enterprise Application Projects: Creating Enterprise Application Projects
- Java Development: Java Application Projects: Opening Projects
- Java EE Development: Enterprise Application Projects: Building Java EE Applications
- Java Development: Enterprise Application Projects: Verifying Java EE Applications
- Java EE Development: Enterprise Application Projects: Deploying Java EE Applications
- Java Development: Java Application Projects: Running Projects
- Server Resources: Java EE Application Servers: Examining Server Log Files
- Java Development: Modifying Project Libraries
- Java Development: Java Classes: Modifying Java Classes: Adding Fields
- Java Development: Java Classes: Opening Java Classes
- Java Development: Other Files: Opening Files
- Java EE Application Servers: Starting Application Servers

- Server Resources: Databases: Starting JavaDB
- Server Resources: Databases: Creating a Java DB Database
- Server Resources: Databases: Connecting to Databases

Task 1 – Implementing the Close Auction Functionality

In this task, you implement the code required to close an auction. Complete the following steps:

1. Open the `AuctionManagerBean` class in the source editor.
2. Locate and implement the following method

```
public void closeAuction(Integer auctionID) throws AuctionException.
```

3. Ensure that you implement the following rules when you use the `closeAuction` method.
   - Close the auction with the primary key field that equals the method input parameter `auctionID`.
   - To close an auction, retrieve it from the database and check whether the status is `OPEN`. If the auction is not `OPEN`, throw an `AuctionException`.
   - To close an auction, change the status to `CLOSED` and merge the auction instance.
4. Build the `AuctionApp` project. Correct any errors you encounter.
5. Verify the `AuctionApp` project.

Task 2 – Testing the Close Auction Functionality

You need to package and deploy the auction system by completing the following steps:

1. Deploy the `AuctionApp` project.
2. Use a browser to load the following URL:

   `http://localhost:8080/AuctionApp/AuctionApp-app-client`
3. You should observe Java Webstart loading the `AuctionApp TestClient` GUI.

   You should observe the following message output on the `AuctionApp Log` window of the GUI:

```
Received hello on startup
```

4. Enter the following string in the AuctionApp TestClient GUI window.

```
addAuction 5000 100 5 car car.jpg 2
```

   You should see the following output:

```
User input: addAuction 5000 100 5 car car.jpg 2
User #2 added an auction with the ID = 6 (Note: Actual ID might vary).
```

5. Use the `Services` tab of the IDE to examine the `AUCTION` table.

   Verify that a new entry has been created for the car auction with an `OPEN` status.

6. Enter the following string in the AuctionApp TestClient GUI window.

```
closeAuction 6
```

   You should see the following output:

```
User input: closeAuction 6
Closed auction 6
```

7. Use the `Services` tab of the IDE to examine the `AUCTION` table.

   Verify that the car auction shows `CLOSED` status.

Task 3 – Implementing the `CreateTimer` Object Functionality

In this task, you implement the code required to create a timer object for each new auction that is created. Complete the following steps:

1. Open the `AuctionManagerBean` class in the source editor.
2. Import the following classes.

```
import javax.ejb.TimerService;
import javax.ejb.Timeout;
import javax.ejb.Timer;
```

3. Add the following injected instance variable declaration to the `AuctionManagerBean` class.

```
@Resource private TimerService timerService;
```

4. Add the `createAuctionTimer` method, as shown in Code 13-1.

Code 13-1 The `createAuctionTimer` Method Template

```
1   private void createAuctionTimer(Date openTime, Date closeTime,
2     Integer auctionID){
3     long duration = 0;
4     // ToDo initialize duration from closeTime - openTime
5     String auctionInfo = “Auction time out: " + auctionID;
6     System.out.println("AuctionManager.createAuctionTimer " +
7       duration + " " +auctionID);
8     // ToDo: create the duration timer using duration and auctionInfo
9   }
```

5. Examine the comment in line 4 of Code 13-1. Initialize the local variable duration to the number milliseconds between the `closeTime` and `openTime` method arguments.
6. Examine the comment in line 8 of Code 13-1. Use the `timerService` instance variable to create a timer object.
7. Modify the `addAuction` method by including a line of code to invoke the `createAuctionTimer` method.
8. Build the `AuctionApp` project. Correct any errors you encounter.
9. Verify the `AuctionApp` project.

Task 4 – Implementing the Timeout Event Handler Method

In this task you implement the code required to capture and process the auction timer timeout event. Complete the following steps:

1. Open the AuctionManagerBean class in the source editor.
2. Add the createAuctionTimer method, as shown in Code 13-2.

Code 13-2 The timeout Method Template

```
1   @Timeout
2   public void timeout(Timer timer) {
3     String infoString = null;
4     // ToDo timer's getInfo method to initialize infoString
5     System.out.println(
6   timeout occurred " +
7       infoString);
8     String auctionIDString = infoString.substring(18);
9     Integer auctionID = new Integer(Integer.parseInt(auctionIDString));
10    System.out.print("Closing Auction " + auctionID);
11    try {
12      // ToDo invoke closeAuction method
13    } catch (Exception e) {
14      System.out.println("AuctionManager.timeout: "
15        "failed to close auction " + auctionID + "  "+ e.getMessage());
16    }
17  }
```

3. Examine the comment in line 4 of Code 13-2. Initialize the local variable infoString to the info object of the timer object.
4. Examine the comment in line 12 of Code 13-2. Invoke the closeAuction method to close the auction.
5. Build the AuctionApp project. Correct any errors you encounter.
6. Verify the AuctionApp project.

Task 5 – Testing the Close Auction Timer Functionality

You need to package and deploy the auction system by completing the following steps:

1. Deploy the `AuctionApp` project.
2. Use a browser to load the following URL:

   `http://localhost:8080/AuctionApp/AuctionApp-app-client`
3. You should observe Java Webstart loading the `AuctionApp TestClient` GUI.

   You should observe the following message output on the `AuctionApp Log` window of the GUI:

```
Received hello on startup
```

4. Enter the following string in the AuctionApp TestClient GUI window.

```
addAuction 5000 100 60 car car.jpg 2
```

   You should see the following output:

```
User input: addAuction 5000 100 5 car car.jpg 2
User #2 added an auction with the ID = 6 (Note: Actual ID might vary).
```

5. Use the `Services` tab of the IDE to examine the `AUCTION` table.

   Verify that a new entry has been created for the car auction.
6. Enter the following string in the AuctionApp TestClient GUI window.

```
addAuctionForSeconds 100 10 60 piano piano.jpg 3
```

   You should see the following output:

```
User input: closeAuctionBeforeTimeout 100 10 60 piano piano.jpg 3
added then closed auction: actionID=8
```

7. Use the `Services` tab of the IDE to examine the `AUCTION` table.

   Verify a new entry has been created for the piano auction. The `STATUS` column should show that the piano auction is open.

   Re-examine the `AUCTION` table after about 60 seconds. The `STATUS` column should show that the piano auction is closed.

Discussion – What needs to be done to cancel the timer associated with an auction if the auction is closed prematurely through the invocation of the closeAuction method?

Exercise 2: Completing Review Questions

In this exercise, you answer questions about the material covered in Module 13 of the Student Guide.

Complete the following questions:

1. Which session bean type can receive timer notifications?
2. Name the annotation required to annotate a timer notification handler method?

Exercise Summary

Discussion – Take a few minutes to discuss what experiences, issues, or discoveries you had during the lab exercise.

- Experiences
- Interpretations
- Conclusions
- Applications

Lab 14

Implementing Security

Objectives

Upon completion of this module, you should be able to:

- Guard the use cases of the auction system
- Complete the review questions

Exercise 1 – Guarding the Use Cases of the Auction System

In this exercise, you work on the `addAuction` use case and the `closeAuction` use case. Both use cases get guarded in such a way that only users who are mapped to a specific EJB security role can perform them. The `addAuction` use case gets guarded by the `user` role and the `closeAuction` use case gets guarded by the `admin` role.

This exercise focuses on applying EJB security.

Use Case Mapping to Groups and Roles

Figure 14-1 on page Lab 14-3 shows the mapping of security groups, roles, and beans in the auction system:

- The `AuctionAdmin` security group is created in the application server's security realm. A user with the login `jim` and the password `007` is added to this group.
- In addition, the `AuctionUser` security group is created in the application server's security realm. A user with the login `dave` and the password `d@v!d` is added to this group.
- For the enterprise beans of the auction system, the `admin` security role and the `user` security role are declared in the `ejb-jar.xml` deployment descriptor file.
- The `closeAuction` method of the `AuctionManager` session bean is declared to map to the `admin` role.
- The `addAuction` method of the `AuctionManager` session bean is declared to map to the `user` role.
- The mapping of roles to groups is declared in the `sun-application.xml` deployment descriptor of the `auctionAppEAR.ear` EAR file.

Note – In the previous exercises, there were no roles declared for the enterprise beans and the enterprise beans' methods were not mapped to security roles. You implement a security group and add users to it by using the application server's administration tools.

Figure 14-1 shows the mapping of security groups, roles, and beans.

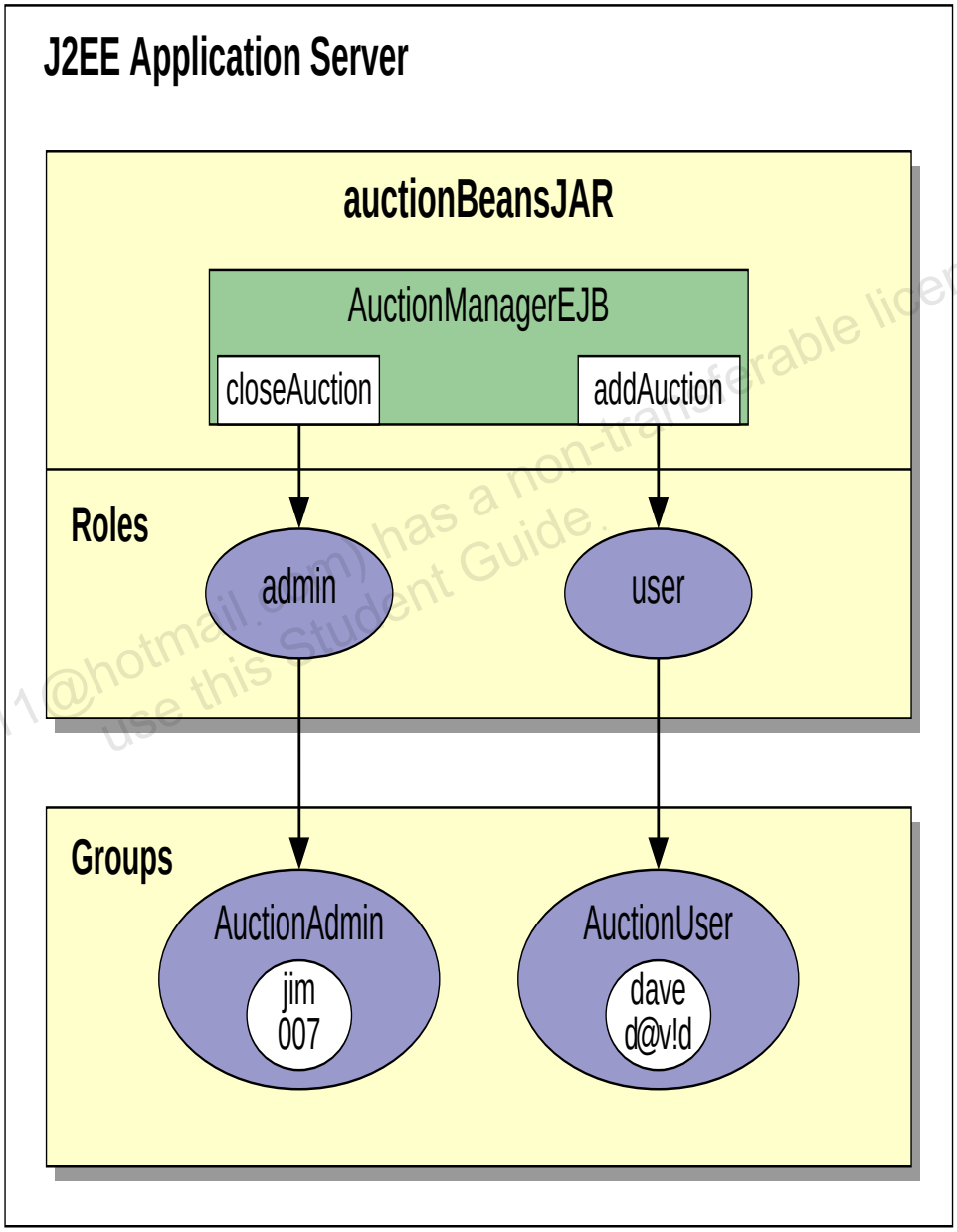

Figure 14-1 Mapping of Security Groups, Roles, and Beans

To provide for the ability to guard the `addAuction` and `closeAuction` use cases of the auction system, you need to perform the following tasks:

This exercise contains the following sections:

- Task 1 – Preparing the Exercise Environment
- Task 2 – Specifying the Security Settings of the `AuctionManager` Session Bean
- Task 3 – Testing the Security Functionality

Tool Reference – Tool references used in this exercise:

- Server Resources: Java EE Application Servers: Undeploying an Application
- Server Resources: Java EE Application Servers: Administrating Security: Adding a File Realm User
- Java Development: Java Application Projects: Opening Projects
- Java EE Development: EJB Modules: Configuring EJB Deployment Descriptors: General Configuration
- Java Development: Java Application Projects: Modifying Project Libraries
- Java Development: Other Files: Deleting Files
- Java Development: Java Classes: Copying Java Classes
- Java EE Development: Enterprise Application Projects: Building Java EE Applications
- Java EE Development: Enterprise Application Projects: Verifying Java EE Applications
- Java EE Development: Enterprise Application Projects: Deploying Java EE Applications
- Java Development: Java Classes: Compiling Java Classes
- Java Development: Java Application Projects: Running Projects

Task 1 – Preparing the Exercise Environment

To prepare the exercise environment:

1. Undeploy the `AuctionApp` project from the Application Server.

Tool Reference – Server Resources: Java Application Servers: Administrating Security: Adding a File Realm User

2. Add auction user to the application server's file realm.
   a. Use the application server's admin console and expand the nodes to view the Edit Realm page:

```
Configuration > Security > Realms > File
```

   b. Press the Manage Users button to obtain a File Users page.
   c. Press the New button to obtain a New File Realm User dialog box.
   d. Enter the following information in the ensuing New File Realm User dialog box.

      User ID: **`jim`**

      Password: **`007`**

      Confirm Password: **`007`**

      Group List: **`AuctionAdmin`**

3. Add auction administrator to the application server's file realm.
   a. Obtain a New File Realm User dialog box.
   b. Enter the following information in the New File Realm User dialog box.

      User ID: **`dave`**

      Password: **`david`**

      Confirm Password: **`david`**

      Group List: **`AuctionUser`**

Task 2 – Specifying the Security Settings of the AuctionManager Session Bean

In this task, you add the declarative security settings for the AuctionManagerBean session bean

1. Open the AuctionManagerBean class in the source editor.
2. Import the following annotations.

```
import javax.annotation.security.DeclareRoles;
import javax.annotation.security.RolesAllowed;
```

3. Add the following annotation to the AuctionManagerBean class. It should follow the @Stateless annotation.

```
@DeclareRoles({"admin", "user"})
```

4. Locate the addAuction method of the AuctionManagerBean class. Annotate it with the following annotation.

```
@RolesAllowed("user")
```

5. Locate the closeAuction method of the AuctionManagerBean class. Annotate it with the following annotation.

```
@RolesAllowed("admin")
```

6. Create the Sun deployment descriptor for the GlassFish server.
   a. Right-click the AuctionApp node in the Projects tab and choose New > Other > GlassFish > GlassFish Deployment Descriptor.

      Accept the default values in the ensuing dialog. The creation of the Sun deployment descriptor causes IDE to create and open the sun-application.xml file in a source editor tab.

   b. Click the XML tab of the sun-application.xml file displayed in the source editor tab.

      Edit the sun-application.xml file to include the two security-role-mapping elements as shown in the following listing:

```
<sun-application>
  <security-role-mapping>
    <role-name>admin</role-name>
    <group-name>AuctionAdmin</group-name>
  </security-role-mapping>
  <security-role-mapping>
    <role-name>user</role-name>
    <group-name>AuctionUser</group-name>
```

```
  </security-role-mapping>
</sun-application>
```

7. Build the `AuctionApp` project. Correct any errors you encounter.
8. Verify the `AuctionApp` project.

Task 3 – Testing the Security Functionality

You need to package and deploy the auction system by completing the following steps:

1. Deploy the `AuctionApp` project.
2. Use a browser to load the following URL:

   `http://localhost:8080/AuctionApp/AuctionApp-app-client`
3. You should observe a login dialog box open.

   Log in as the auction user `dave`, with password `david`.
4. You should observe Java Webstart loading the `AuctionApp TestClient` GUI.

   You should observe the following message output on the `AuctionApp Log` window of the GUI:

```
Received hello on startup
```

5. Enter the following string in the AuctionApp TestClient GUI window.

```
addAuction 5000 100 5 car car.jpg 2
```

   You should see the following output:

```
User input: addAuction 5000 100 5 car car.jpg 2
User #2 added an auction with the ID = 6 (Note: Actual ID might vary).
```

6. Use the `Services` tab of the IDE to examine the `AUCTION` table.

   Verify that a new entry has been created for the car auction with an `OPEN` status.
7. Enter the following string in the AuctionApp TestClient GUI window.

```
closeAuction 6
```

   You should get an authentication exception displayed in the application server log window.
8. Try other use cases, such as `addItem`.

   You should be able to invoke all other use cases.
9. Close the client application and log in again as auction admin `jim` with password `007`.
10. Close the auction you created.

    The auction system should close the auction. Verify using the `Services` tab and examining the AUCTION table. The status of the car auction should indicate closed.

Exercise 2: Completing Review Questions

In this exercise, you answer questions about the material covered in Module 14 of the Student Guide.

Complete the following questions:

1. List the identities recognized in a Java EE application?
2. Name the annotation used to declare roles in a Java EE application?

Exercise Summary

Discussion – Take a few minutes to discuss what experiences, issues, or discoveries you had during the lab exercise.

- Experiences
- Interpretations
- Conclusions
- Applications

Lab 15

Using EJB Technology Best Practices

Objectives

Upon completion of this lab, you should be able to:

- Complete the review questions

Exercise 1: Completing Review Questions

In this exercise, you answer questions about the material covered in Module 15 of the Student Guide.

Complete the following questions:

1. Which EJB component is best suited to model the business process?
2. Which component is best suited to model business data?