Aula Matriz S.A.S.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Developing Web Services Using Java Technology

Activity Guide - Solaris

DWS-4050-EE6 Rev A

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide

D65185GC11
Edition 1.1
September 2010
D69079

ORACLE®

Aula Matriz S.A.S.

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Disclaimer

This document contains proprietary information, is provided under a license agreement containing restrictions on use and disclosure, and is protected by copyright and other intellectual property laws. You may copy and print this document solely for your own use in an Oracle training course. The document may not be modified or altered in any way. Except as expressly permitted in your license agreement or allowed by law, you may not use, share, download, upload, copy, print, display, perform, reproduce, publish, license, post, transmit, or distribute this document in whole or in part without the express authorization of Oracle.

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.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Aula Matriz S.A.S.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

This page intentionally left blank.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Aula Matriz S.A.S.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

This page intentionally left blank.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Aula Matriz S.A.S.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Contents

Aula Matriz S.A.S.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Aula Matriz S.A.S.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Aula Matriz S.A.S.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Aula Matriz S.A.S.

Oscar Hernando Olivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Aula Matriz S.A.S.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Oscar Hernando Olivia Aragon (olivia@hotmail.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Aula Matriz S.A.S.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Aula Matriz S.A.S.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Aula Matriz S.A.S.

Lab Preface

Workbook Goals

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

- Describe how web services frameworks can be used to deploy and consume services.
- Use the JAX-WS technology to deploy and consume web services.
- Use the JAX-RS technology to deploy and consume web services.
- Deploy web services that also leverage other Java^{TM} Platform, Enterprise Edition (JavaEE)technologies, such as the web container infrastructure, Enterprise Java Beans, or the Java Persistence API.
- Apply best practices and design patterns when designing and deploying web services using either JAX-WS or JAX-RS technologies.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Aula Matriz S.A.S.

Conventions

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

Icons

Self-check - Identifies self-check activities, such as matching and multiple-choice.

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.

Tool Reference Guide - Indicates the entry in the NetBeans Tool Reference Guide that describes how to achieve the task at hand. If you are reading the PDF version of this document, the icon is a hyperlink: click on it, and the Tool Reference Guide will open to the proper page – assuming that this document and the Tool Reference Guide are found in the same folder on your system.

Important - Important

Typographical Conventions

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

Use `ls -al` to list all les.

```
system# You have mail.
```

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Aula Matriz S.A.S.

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 les 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^{TM} 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:

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Aula Matriz S.A.S.

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^{TM}Operating Environment is different from a command used in the Microsoft Windows platform, both commands are shown; for example:

  If working in the Solaris Operating Environment

  ```
  %cd $SERVER_ROOT/BIN
  ```

  If working in Microsoft Windows

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

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Aula Matriz S.A.S.

Setup

Objectives

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

- Configure the environment that will be needed for all the exercises in the course.
- Run JUnit tests on the initial model logic for both the Traveller and Auction domains.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Aula Matriz S.A.S.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Exercise 1 – Configuring Environment for All Labs

In this exercise, we set up the environment that will be required to complete all the labs in this course. During the course, we will have opportunity to develop, deploy, and test services that manipulate persistent objects in two different domains, airline reservations (the Traveller application) and online auctions (the Auction application).

In order to do so, we are going to need to have a back-end database configured, and available to NetBeans, our applications, and the GlassFish application server.

We are also going to need a framework to test web services, and the configuration of several identities for use in clients that interact with the application server (to test security configuration settings.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Task 1 – Create Databases

We are going to need two databases for the development projects associated with the course: one for the Traveller application used during lectures, and the other for the Auction application you will developer as you complete the different projects presented during the course.

To create a database,

- Select the "Services" tab in the NetBeans desktop.
- Right-click on the "JavaDB" node, and select "Create Database" from the pop-up menu.

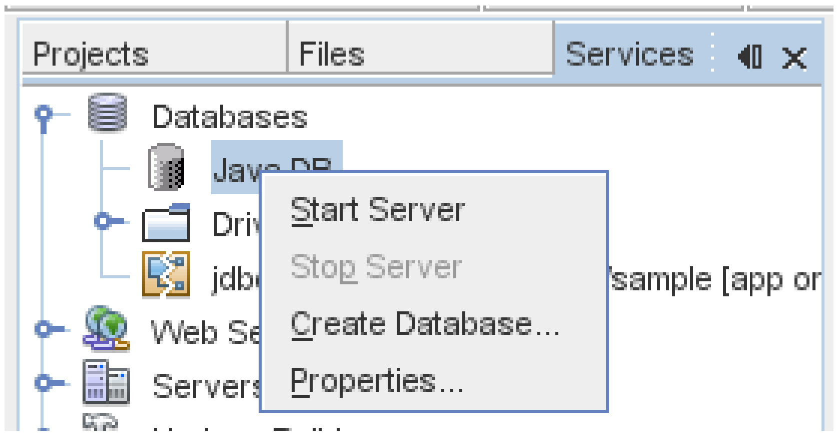
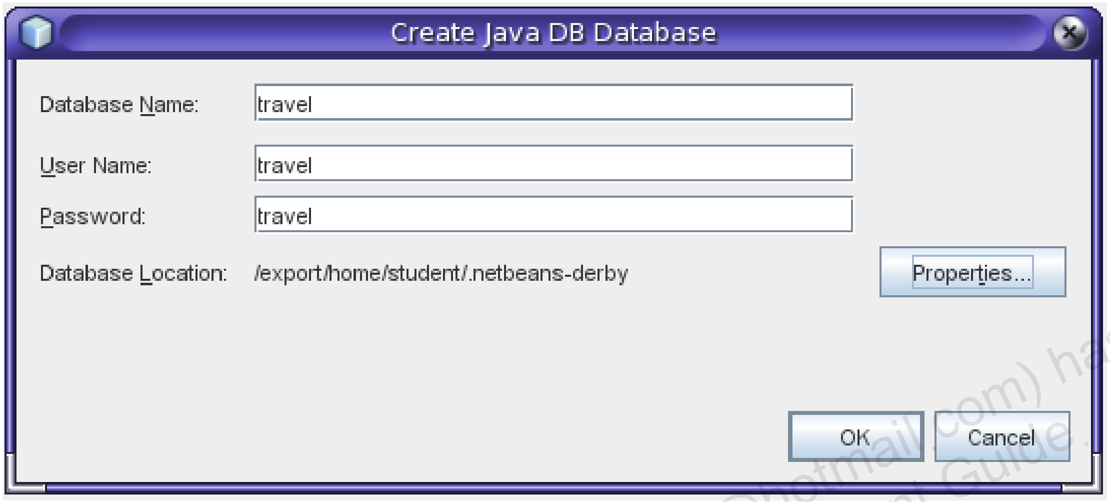


Figure 1: Getting to the `Create a Database` option in NetBeans.

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Aula Matriz S.A.S.

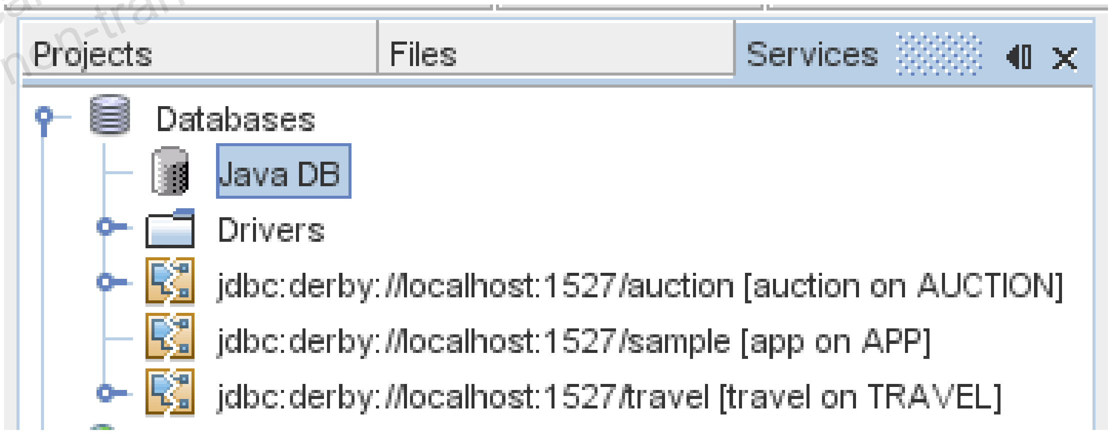
- Select the menu item "Create Database..."
- Enter database name and username/password combination to use to maintain the new database. You should use "travel" for all three of "username", "password", and "database name", as shown in Figure 2



Database Name: travel
User Name: travel
Password: travel
Database Location: /export/home/student/.netbeans-derby

Figure 2: Creating the Databases

- You should create a second database, using "auction" for all three fields.
- At the same time the two databases are created, NetBeans should also create connections to the two databases under the `Databases` node in the Services tab, as show in Figure 3.



```
Databases
  Java DB
  Drivers
  jdbc:derby://localhost:1527/auction [auction on AUCTION]
  jdbc:derby://localhost:1527/sample [app on APP]
  jdbc:derby://localhost:1527/travel [travel on TRAVEL]
```

Figure 3: Connections to Databases

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.
Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Aula Matriz S.A.S.

Note the difference in the icons that represent the two connections:

represents an active connection

represents a connection that is currently down

You can establish or shut down connections here by right-clicking on the connection to affect, then selecting from among the options that appear in the pop-up menu. Once you have established one of these connections, you can also use this node to browse the content of the associated database.

```
Server Resources
→ Databases
→ Creating a JavaDB Database
```

Task 2 – Create Database Connections – Optional

When NetBeans creates new databases, it also pre-configures JDBC connections to the databases it creates into NetBeans. On the other hand, if the databases listed already exist on the host computer, NetBeans will report an error during the dialog that should attempt to create the new database. In second case, it won't pre-configure a new JDBC connection to that database.

If NetBeans reports that the database that you wish to create already exists, but it doesn't show up as a child of the Database node, it may be that NetBeans is missing a NetBeans-based connection to the particular database:

1. To simply create a new NetBeans connection to an existing database, right-click on the `Databases` node in the `Services` panel in NetBeans, then select `Create New Connection...` from the context menu that results from the right-click.

```
Server Resources
→ Databases
→ Connecting to Databases
```

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Aula Matriz S.A.S.

Task 3 – Configure JDBC Drivers for Application Use

The examples used throughout the course are available as NetBeans projects in your `labFiles` folder. The list of projects available includes:

- `projects/TravellerEntities`
  This project contains the domain objects listed for the Traveller project. The project contains both a set of domain types implemented as persistent objects using JPA, and a set of data access types (DAOs), to hide the dependency on JPA as much as possible.

- `projects/TravellerEJBs`
  This project reimplements the DAOs listed in the TravellerEntities project so that they are all implemented as session beans. This provides better encapsulation of the JPA machinery within the session beans.

- `examples/jaxrs/ResourceManagers`
  This project contains all the JAXRS examples used in the course that are implemented as POJOs, and deployed within a simple JVM.

- `examples/jaxrs/JavaEEJAXRS`
  This project contains all the JAXRS examples used in the course but implemented as JavaEE components, or deployed to a JavaEE container.

- `examples/jaxws/Managers`
  This project contains all the JAXWS examples used in the course that are implemented as POJOs, and deployed within a simple JVM.

- `examples/jaxws/JavaEEJAXWS`
  This project contains all the JAXWS examples used in the course but implemented as JavaEE components, or deployed to a JavaEE container.

- `projects/AuctionApp`
  This project contains all the domain types listed for the Auction project. The project contains both a set of domain types implemented as persistent objects using JPA, and a set of data access types (DAOs), to hide the dependency on JPA as much as possible.

- `projects/AuctionEJBs`
  This project contains all the domain types listed for the Auction project. The project contains both a set of domain types implemented as persistent objects using JPA, and a set of data access types (DAOs) implemented as stateless session beans, to hide the dependency on JPA as much as possible.

All of these projects depend on the presence of a non-standard NetBeans library called `DerbyJDBC`, which includes the JDBC drivers required to talk to JavaDB,

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Aula Matriz S.A.S.

the persistence engine implemented in Java and distributed with NetBeans and GlassFish.

Although the JDBC drivers are actually available with GlassFish, NetBeans sometimes is missing a library entry for this particular library. If so, when you try any of the projects above, NetBeans will complain about this missing library.

You can add the missing entry for the `DerbyJDBC` library to NetBeans yourself:

1. On the NetBeans main menu, click on `Tools`, then `Plugins`.
2. On the plugins window, click on `New Library...`, to add a new library. A new dialog pops up, to create a new library.
3. Enter `DerbyJDBC` as the name of the new library. Figure 5 shows what this first dialog might look like.
4. Click OK, to submit this dialog. The original dialog, which lists all libraries, should now include an entry for `DerbyJDBC`.
5. Ensure that `DerbyJDBC` is the selected library, then click on the button labeled `Add Jar/Folder...` to a add a new jar file to this new library. A new dialog will pop up, to allow you to browse for the appropriate file. Select the JavaDB JDBC driver jar file, which can be found in

   ```
   sges-v3/javadb/lib/derbyclient.jar
   ```

   within the student's home directory.
6. Click on the `Add Jar/Folder` button to submit this dialog.
7. Click on `OK` to close the original `Library Manager` dialog.

```
Server Resources
→ Databases
→ Adding JDBC Drivers
```

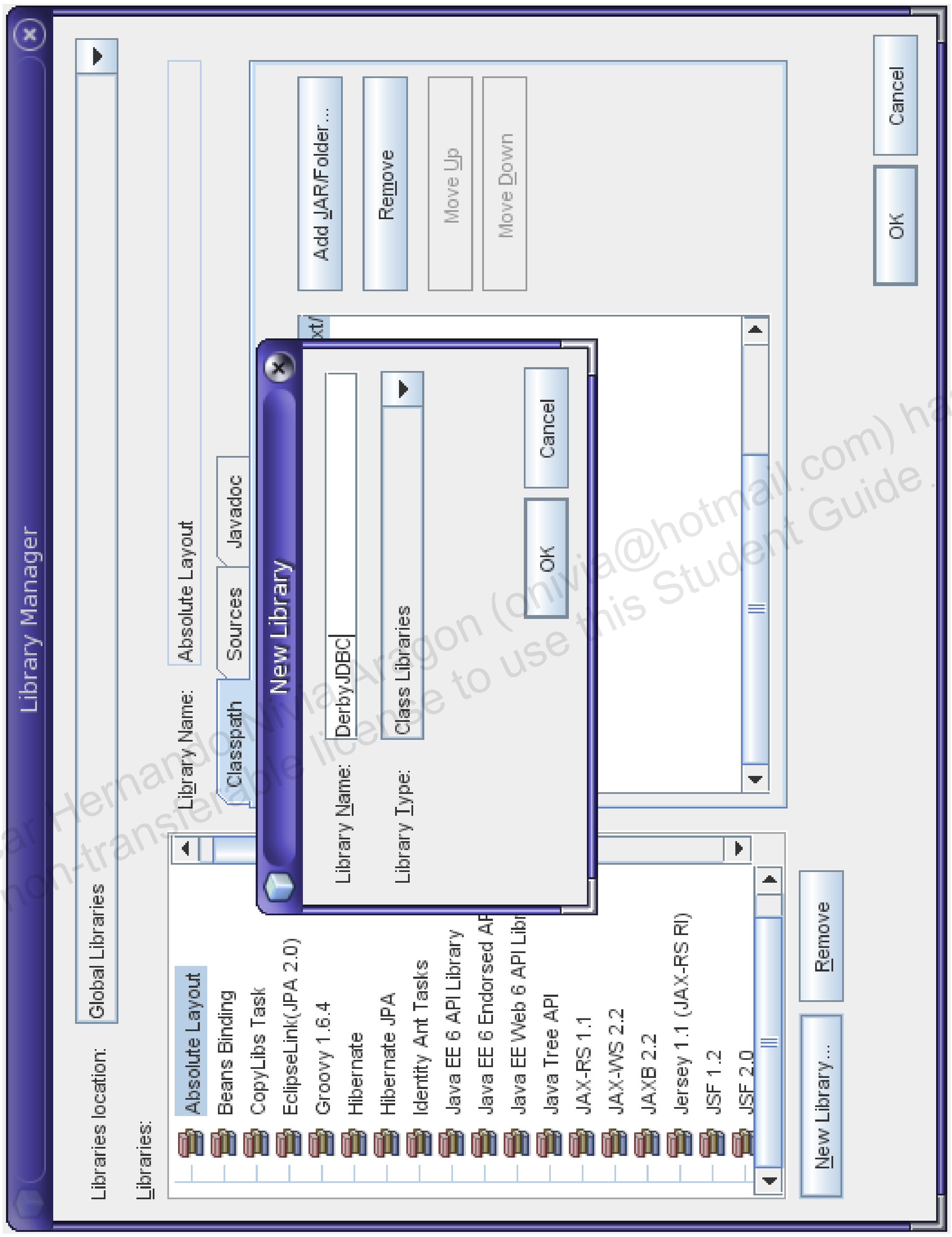
Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Aula Matriz S.A.S.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.



Library Name: DerbyJDBC

Library Type: Class Libraries

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Figure 4: Dialog to Create New `DerbyJDBC` Library

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Aula Matriz S.A.S.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

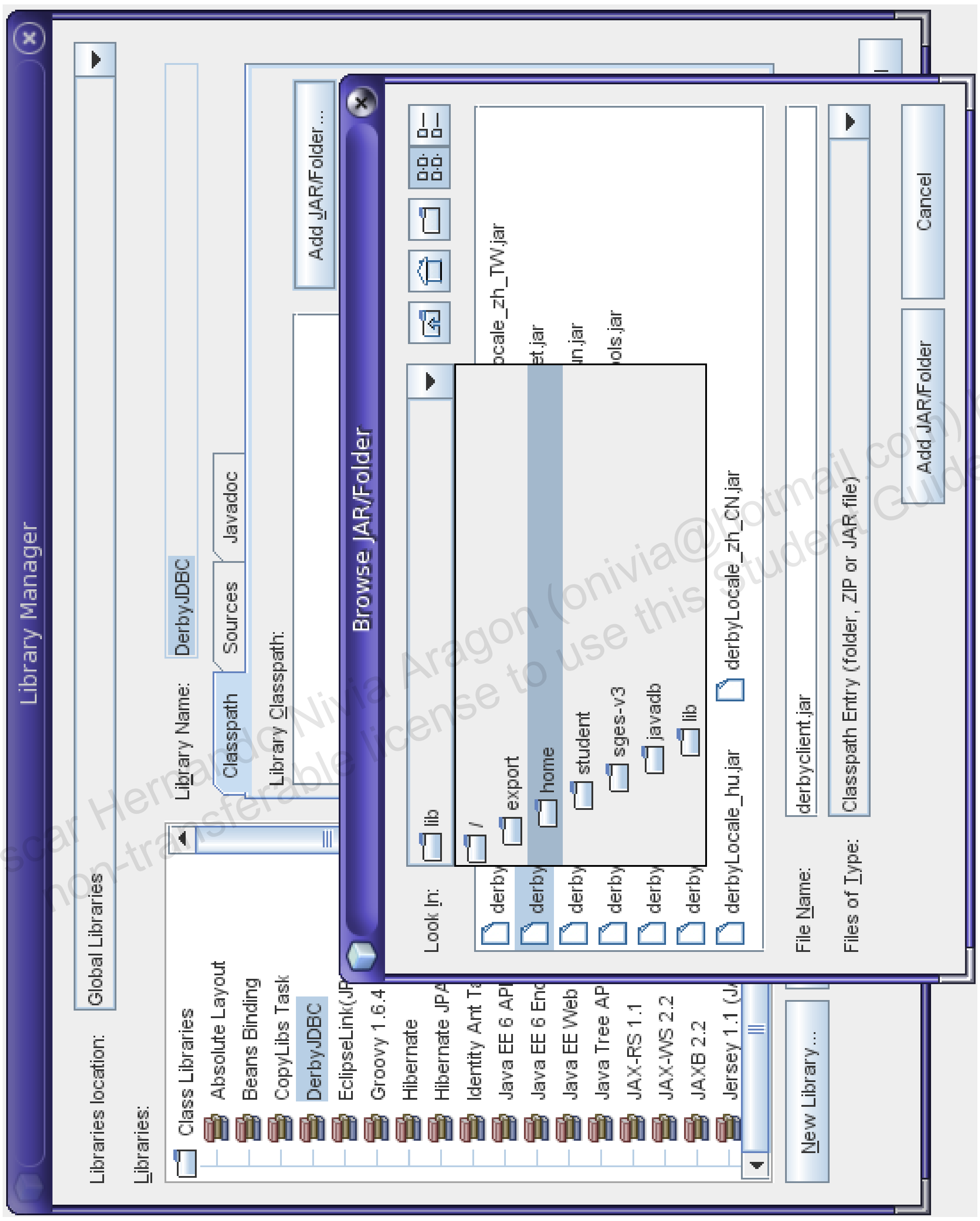


Figure 5: Dialog to Create New `DerbyJDBC` Library

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Aula Matriz S.A.S.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Exercise 2 – Running JUnit Tests on Domain Classes for Traveller Domain

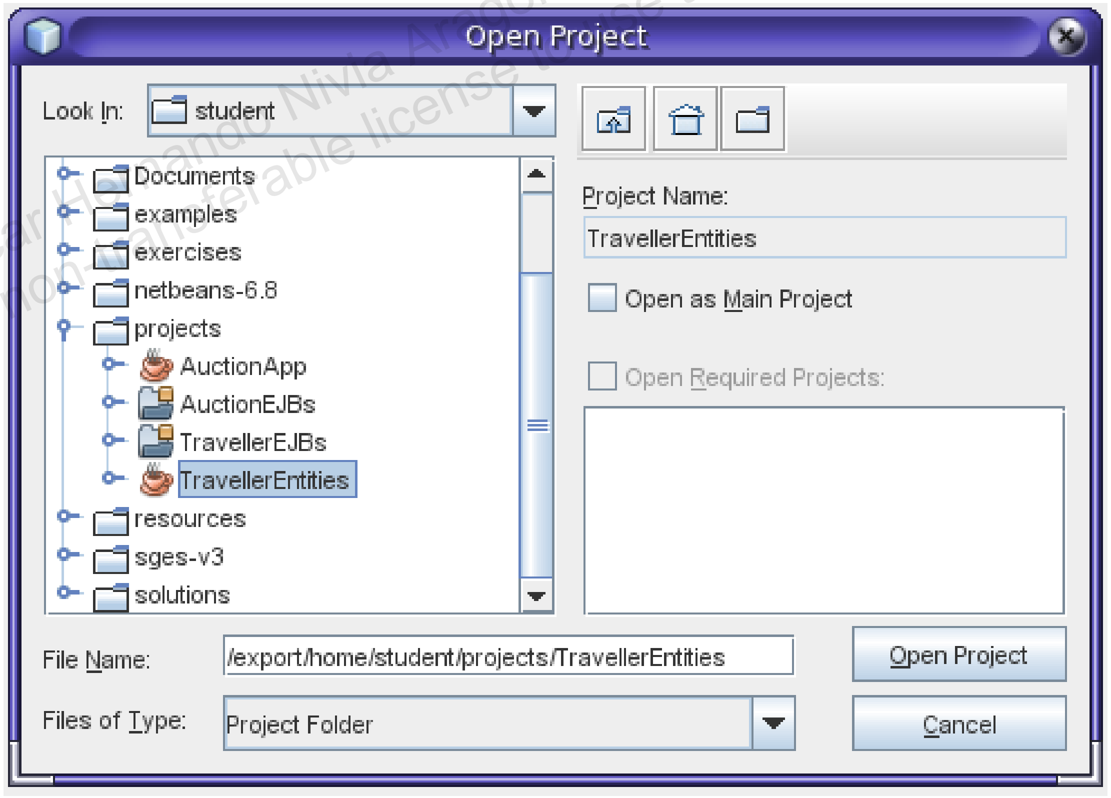
You can see the domain types that we provide for the Traveller project "in action" by running the JUnit tests that we provide within the TravellerEntities project. These JUnit tests exercise the DAOs provided within the project, which then add and manipulate persistent objects of the types in the project.

To run the JUnit tests for the Traveller project:

- Open the `TravellerEntities` project: click on the `File` menu, then choose `Open Project...` You will see dialog like the one shown in Figure 6 – you will have to drill down into the projects folder, to see the NetBeans projects found within.

```
Java Development
→ Java Application Projects
→ Opening Projects
```

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.



Project Name: TravellerEntities

File Name: /export/home/student/projects/TravellerEntities

Files of Type: Project Folder

Figure 6: Opening Projects

Aula Matriz S.A.S.

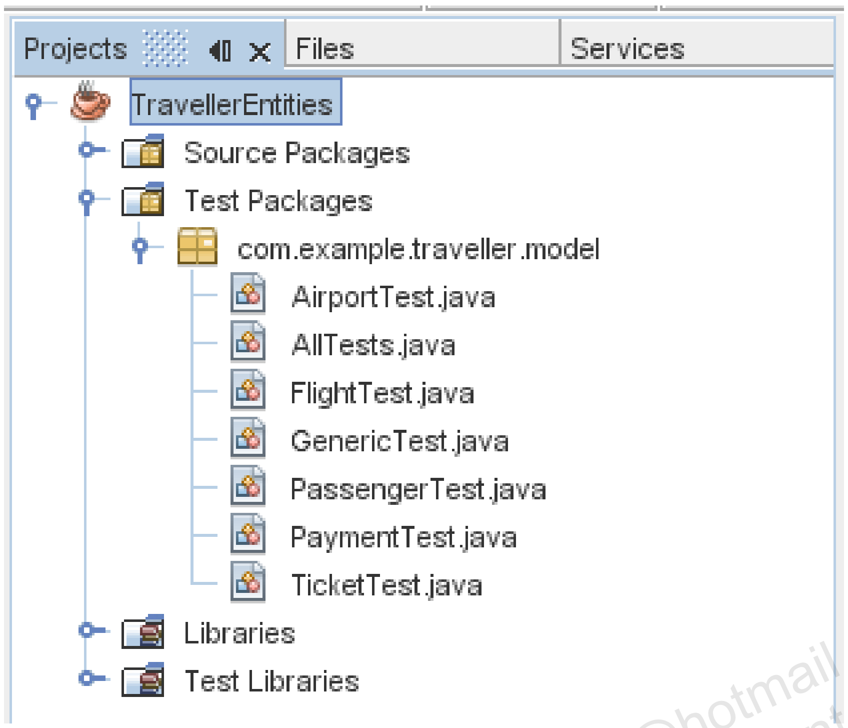


Figure 7: `TravellerEntities` Project

- Navigate to the `Test Packages` node in the `Projects` tab, within the `TravellerEntities` project you just opened. Figure 7 illustrates what the project structure will look like.
- Right click on any of the test classes within the `Test Packages` node in the project tab, then choose `Test File`.

```
Java Development
→ Java Classes
→ JUnit Test Classes
```

You should leave this project open, after running the JUnit tests, as other projects will require this one.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.
Oscar Fernando Nuñez Aragon (oriiia@hotmail.com) has a non-transferable license to use this Student Guide.

Aula Matriz S.A.S.

Exercise 3 – Running JUnit Tests on Domain Classes for the Auction Domain

You can see the domain types that we provide for the Auction project "in action" by running the JUnit tests that we provide within the `AuctionApp` project. These JUnit tests exercise the DAOs provided within the project, which then add and manipulate persistent objects of the types in the project.

You should leave this project open, after running the JUnit tests, as other projects will require this one.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Aula Matriz S.A.S.

Exercise Summary

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

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Aula Matriz S.A.S.

Course Projects

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Aula Matriz S.A.S.

The Traveller Project

Traveller Project – Use Cases – Projects The Traveller application is the project from which the course draws all of its examples. The Traveller application supports a set of use cases that would necessary to provide an online airline reservation web site, as illustrated by Figure 8.

addAirport Adds a new airport to the system.

findAirport Finds an airport known to the system, by airport code, or by airport name.

findNeighbors Find all airports that can be reached from a given airport in one flight.

findFlights Find all (direct) flights between two airports.

makeReservation Make a reservation for a customer on a flight. In this system, a round-trip flight requires two reservations, one for each one-way trip.

addFlight Add a flight between the two airports specified to the system.

The uses cases listed above are just a sample of the uses cases that such an application should support – but they are enough to describe all the examples used throughout the course.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.
Oscar Hernando Nivia Aragon (olivia@hotmail.com) has a non-transferable license to use this Student Guide.

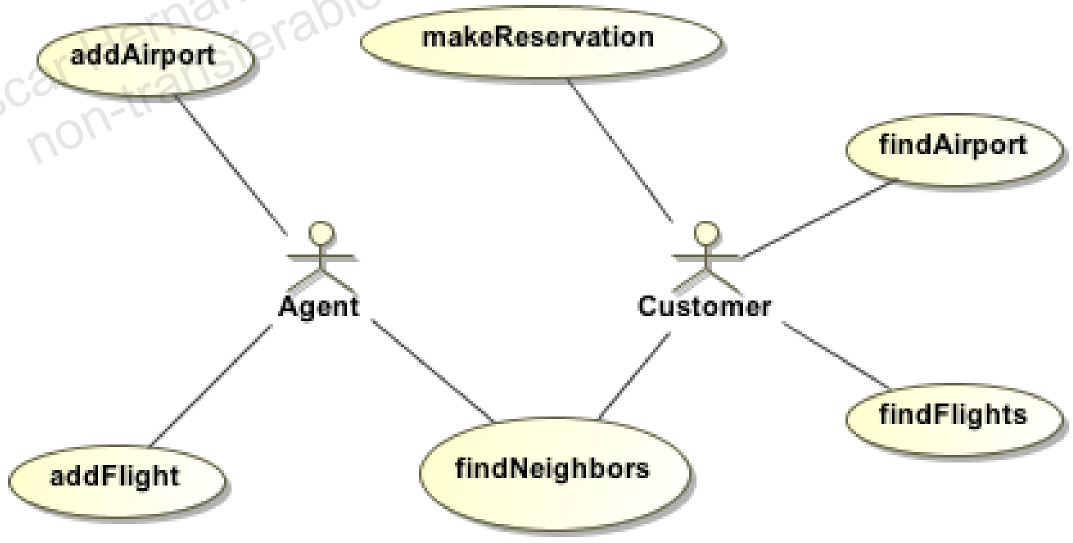


Figure 8: Use Cases for Traveller Project

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Aula Matriz S.A.S.

Traveller Project – Domain – Projects Figure 9 is a sketch of the domain types used within the Traveller project to capture the information necessary to implement the uses cases described in Figure 8. In the design used by the course, all of these types are persistent – and all of these are implements as JPA entity classes, to minimize the effort required to manage their persistence.

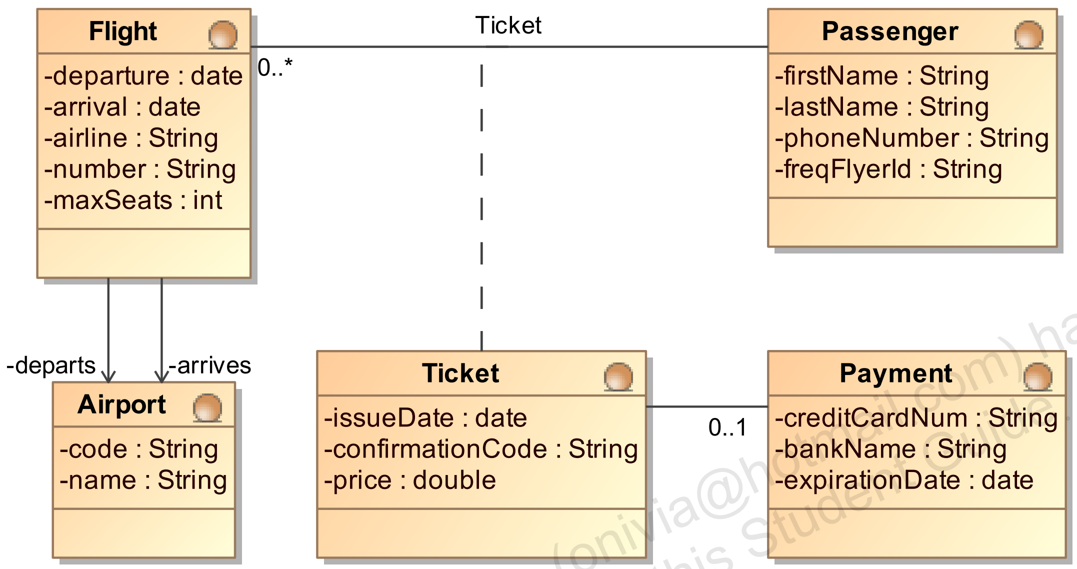


Figure 9: Domain Types for Traveller Project

Traveller Project - DAOs Figure 10 presents a UML diagram of a number of *Data Access Object* classes, which try to hide the JPA machinery that is used to manage the persistent instances used by the application. In the implementation offered for the examples, these DAO classes will be implemented two ways: as simple Java classes, and as Enterprise Java beans. The first implementation is less successful than the second one at hiding the JPA machinery: absent EJBs, the application logic must manage persistence contexts and transactions explicitly – and this may need to be exposed all the way to business methods...

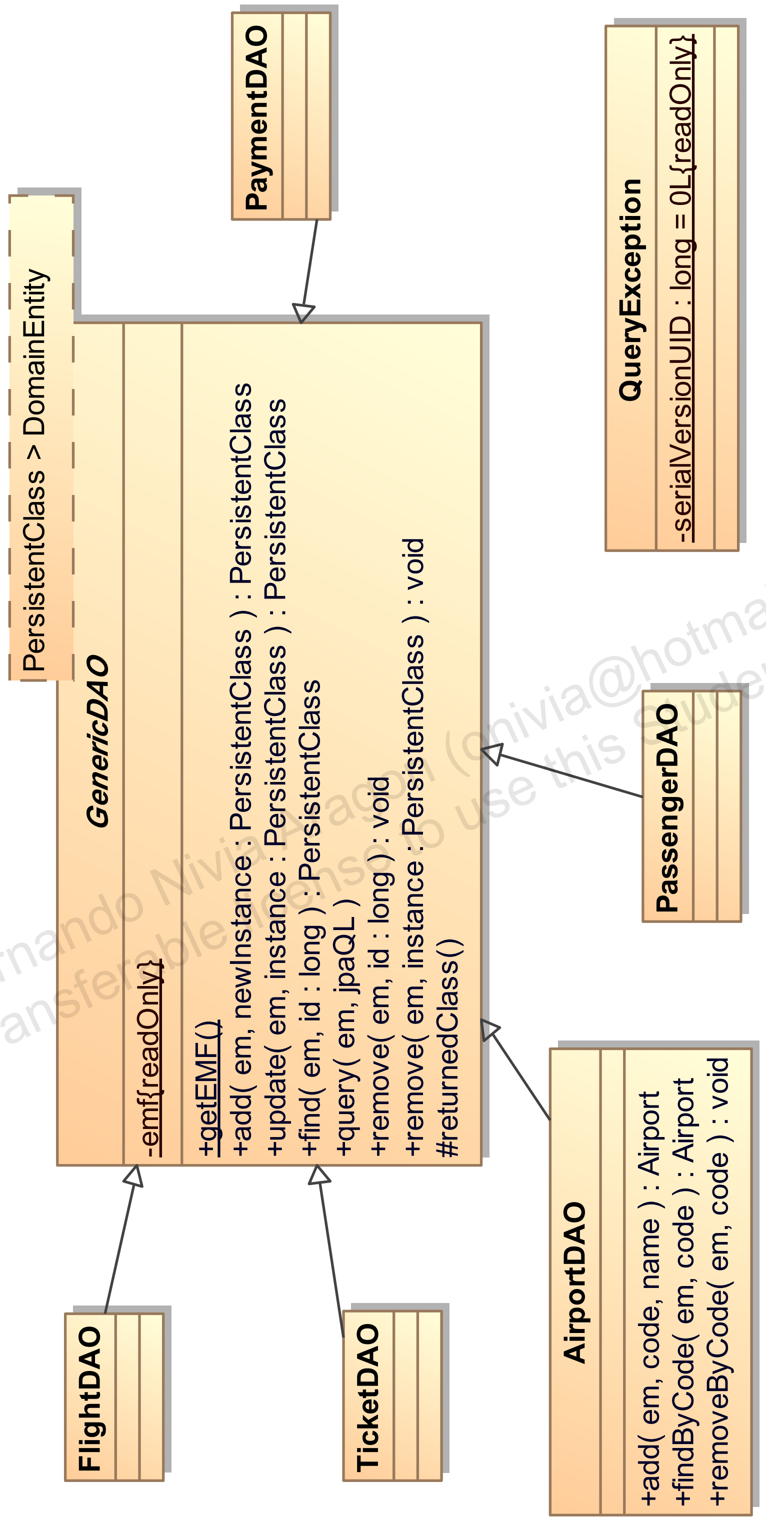
Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Oscar Fernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Aula Matriz S.A.S.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.


PaymentDAO
QueryException
-serialVersionUID : long = 0L{readOnly}
PersistentClass > DomainEntity
GenericDAO
-emf{readOnly}
+getEMF()
+add(em, newInstance : PersistentClass) : PersistentClass
+update(em, instance : PersistentClass) : PersistentClass
+find(em, id : long) : PersistentClass
+query(em, jpaQL)
+remove(em, id : long) : void
+remove(em, instance : PersistentClass) : void
#returnedClass()
PassengerDAO
FlightDAO
TicketDAO
AirportDAO
+add(em, code, name) : Airport
+findByCode(em, code) : Airport
+removeByCode(em, code) : void

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Figure 10: DAO Classes for the Traveller Project

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Aula Matriz S.A.S.

The Auction Project

Auction Project - Use Cases

The Auction application supports the set of use cases necessary to provide an online auction web site, as illustrated by Figure 11:

`login` All users of the Auction system must first log on to the system, before they can invoke other operations. To log on, all users must authenticate themselves (perhaps by providing a userid and a password). This use case includes either of the `addUser` or `findUser` use cases.

It might be important to distinguish between sellers and bidders, since there are tasks in the Auction system that only one kind of user may request.

`addUser` Adds a new user to the system database. This use case is invoked by the `login` use case, when a new user accesses the Auction system for the first time. In addition to the user ID and password already supplied, the system asks the user for additional personal information: name, address (both location and email). Optionally, the user will be able to provide credit card information to be used as a default method of payment.

`findUser` Finds an existing user in the system database. This use case is in-

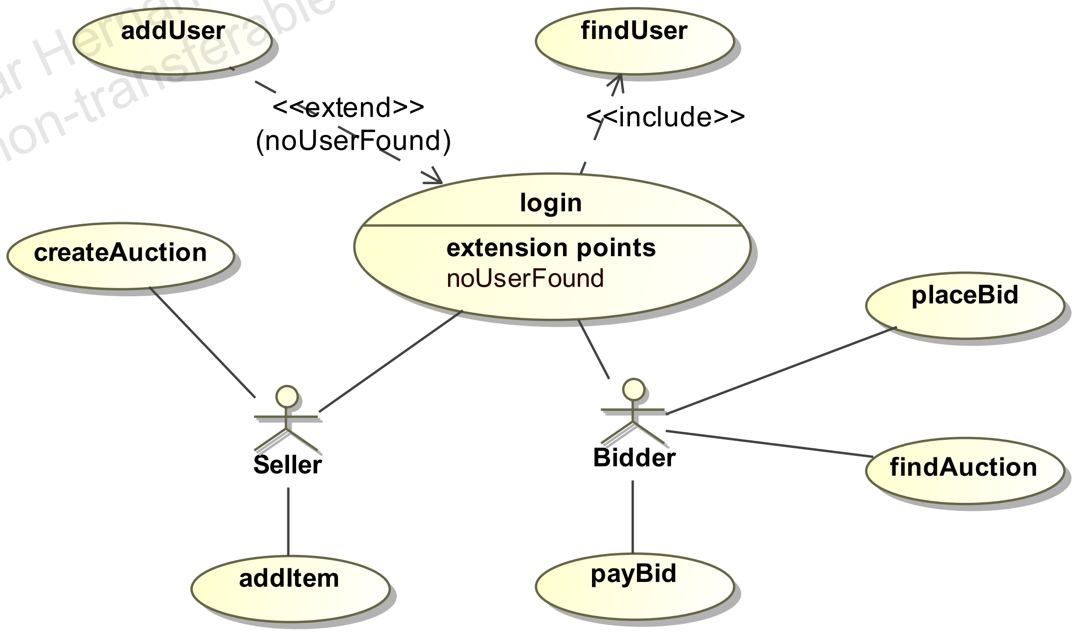


Figure 11: Use Cases for Auction System

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.
Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Aula Matriz S.A.S.

voked by the `login` use case, when an existing user returns to the Auction system.

`addItem` Sellers are allowed to create items to be auctioned off. The item to be auctioned and the auction for the item are considered to be independent from each other. This allows a seller who has multiples of a given item to hold separate auctions for each copy, while reusing the same item for each auction. Items hold information, such as the name for the item, a description, and a picture.

`createAuction` Sellers are allowed to create auctions, to auction items off. Each auction holds information, such as the item being auctioned, the initial bid price, the date the auction starts and the date it will end (or how long the auction will last), and a list of all bids placed on it.

The Auction system should also allow the seller to create multiple auctions for a number of copies of the same item conveniently. This can be achieved by allowing the seller to create a first auction, and then allowing the seller to create "another just like it", or by allowing the seller to specify the number of copies of an auction to create, as part of the process of creating each auction.

`placeBid` Bidders are allowed to bid on auctions. Given an auction, the bidder is allowed to enter a bid on that auction. The Auction system should not allow a bidder to place a bid on an auction that is lower than the current bid price for that auction. The Auction system should also allow the user to find out what the current bid price for the auction is, or even notify the user automatically when the current bid price for the auction changes, or when the bidder is no longer the current high bidder for the auction.

The bidder should be able to select a collection of auctions they are interested in bidding on, and to allow the bidder to place bids on any of the auctions in that collection. In this case, the Auction system should present the bidder with the current bid price for each of the auctions in the collection, plus a running total of the amount of money the bidder is currently bidding on all auctions in the list, taken together. Also, bidders should be able to find out what the current bid prices for all auctions in the list are, or be notified automatically when the current prices change, or when the bidder is no longer the high bidder for any of the auctions in the collection.

`findAuction` Users must be able to provide criteria to be used to retrieve a collection of bids in which the user might be interested. The user must also be able to narrow down the results of a previous query.

These query operations must be supported while other users are using

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Aula Matriz S.A.S.

these same auctions. That is, users must be allowed to find auctions of interest while other users do the same, or bid on some of the same auctions.

`payBid` Bidders must be able to pay for their winning bid on an auction, once the action is over. The Auction system offers to use the credit card information the user entered when the user first registered with the system, although it also allows the user to provide an alternative means of payment. The system should also notify the seller of that auction, when the winning bidder actually pays for that auction.

Auction Project - Domain The business model for the Auction system represents all the information necessary to support the use cases listed previously. Figure 12 illustrates the following abstractions, as a starting point to build the business tier for this application:

`Auction` Each instance represents an auction. It is
responsible for the dates the auction starts and ends,
the initial and winning bid price, and a list of all bids placed on this auction.

`Item` Each instance represents an item that can be auctioned off. It is responsible for information about each item, such as its name, description, or a picture of the item.

`Bid` Each instance represents a bid that a user has placed on an auction. It is responsible for information about the bid, such as the maximum amount the bidder was willing to bid on this bid, the time the bid was placed, and the bidder who placed the bid.

`User` Each instance represents a user of the Auction system. It is responsible for information about each user, such as the user's name, address, email, or credit card information.

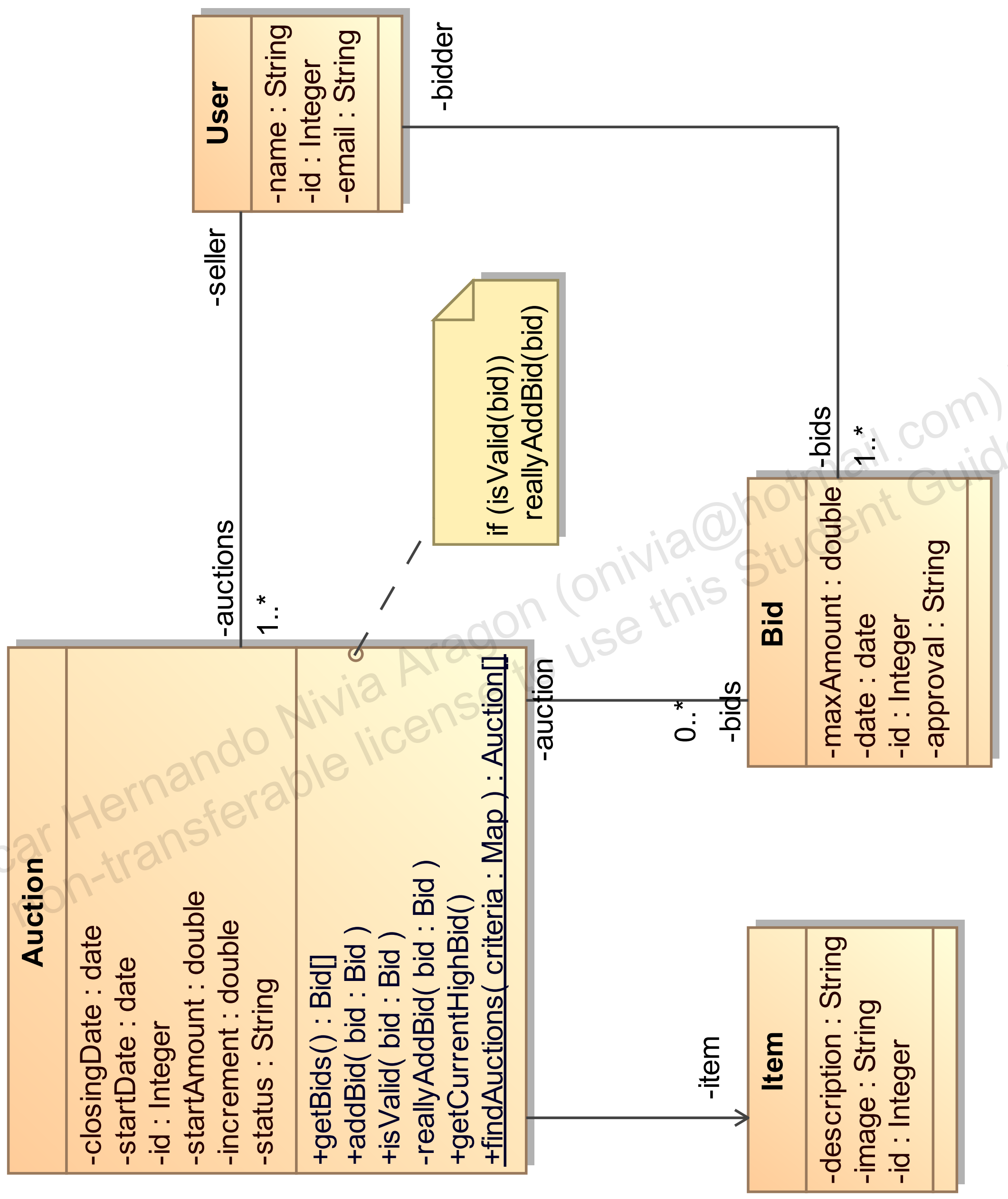
Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Aula Matriz S.A.S.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.



Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Figure 12: Domain Types for Auction System

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Aula Matriz S.A.S.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Level I

High-Level Labs

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Aula Matriz S.A.S.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Lab 1

Introduction to Web Services

Objectives

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

- Describe the characteristics of a web service.
- Describe the advantages of developing web services within a JavaEE container.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Aula Matriz S.A.S.

Exercise 1 – Describing Web Services

1. Which of the following are characteristics of a web service? (Choose all that apply)

   (a) Tightly-coupled

   (b) Loosely-coupled

   (c) Interoperable

   (d) Platform-specific

2. Indicate, for each of the following, whether they are a characteristic of traditional RPC solutions, or web services. (Write either "RPC" or "Web Services" on the line next to each)

   Local to an enterprise ________________

   Message-driven ________________

   Procedural ________________

   Firewall-friendly ________________

   Variable transports ________________

3. Mark in each cell in the table whether the web services technology listed exhibits the characteristic listed:

	SOAP	**REST**
Use XML		
Has Formal Definition		
Relies on HTTP		

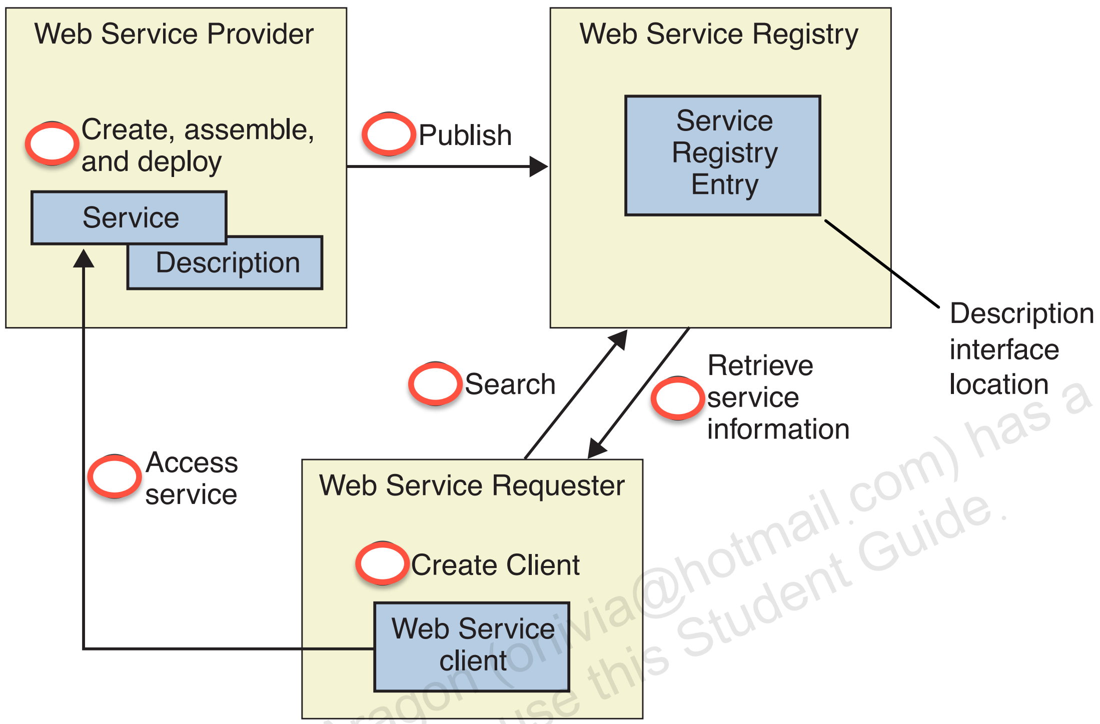
Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Aula Matriz S.A.S.

4. Fill in the red ovals with numbers, indicating the order in which the steps illustrated in this figure are executed:



5. Select the features that correspond to support that the EJB component model offers web services developers. (Choose all that apply).

   (a) Programmatic transaction control.

   (b) Declarative transaction control.

   (c) Scalability through pooling.

   (d) Availability through robustness of Java implementation/runtime.

6. Which of the choices below constitutes deployment choices that are open to the web services deployer? (Choose all that apply).

   (a) Standalone Java applications on any JVM, because of web server machinery built-into JAX-WS and JAX-RS runtime.

   (b) Servlet endpoints deployed to a web container.

   (c) EJB service endpoints.

   (d) JMS-based runtimes, because the choice of a JMS transport is required by the web services specifications.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Aula Matriz S.A.S.

7. Web services and their clients can interact with each other regardless of the platform on which they are running. This is possible in a SOA by means of (choose one):

   (a) Compatibility

   (b) Interoperability

   (c) Accessibility

   (d) Security

8. Which is the only characteristic of a service that a requesting application needs to know about in SOA? (Choose one).

   (a) Private interface

   (b) Protected interface

   (c) Public interface

   (d) Package interface

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Aula Matriz S.A.S.

Lab Summary

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

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Aula Matriz S.A.S.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Aula Matriz S.A.S.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Lab 2

Using JAX-WS

Objectives

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

- Create simple web services using JAX-WS:
  - Code-first, starting from Java classes
  - Contract-first, starting from WSDL descriptions
- Deploy web service providers using just JavaSE.
- Create and Run web service clients.

Oscar Hernando Nivia Aragon (olnivia@hotmail.com) has a non-transferable license to use this Student Guide.

Aula Matriz S.A.S.

Exercise 1 – Understanding JAX-WS

1. Identify the correct tool to be used for generating JAX-WS artifacts in each of the following cases:
   (a) From a URL of a WSDL file
   (b) From a Java implementation source file
   (c) From a Java class file
2. Which is the annotation used to describe an argument of a web service operation. (Choose one).
   (a) `@WebResult`
   (b) `@WebService`
   (c) `@WebMethod`
   (d) `@WebParam`
3. Building web services according to the "contract-first" approach is considered a best practice because:
   (a) Writing WSDL descriptions first is more maintainable, because WSDL descriptions are more concise.
   (b) Writing Java code first is more flexible, because the designer can capture characteristics of the service in Java code that cannot be captured in WSDL.
   (c) Writing Java code first is safer, because the developer can capture constraints in Java code that cannot be captured in WSDL.
   (d) Writing WSDL descriptions first is safer, because the developer can capture constraints in WSDL descriptions that cannot be captured in Java code.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable licence to use this Student Guide.

Aula Matriz S.A.S.

Exercise 2 – Create, Deploy, and Test a JAX-WS Web Service Code-First

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

In this exercise, you're going to create, deploy, and test a simple web service for the Auction application. You will implement this web service using a "code-first" approach, implementing the Java class that will be the service provider first, then turning it into a web service through the introduction of JAX-WS annotations. Once you have your web service ready, you will deploy it to a stand-alone Java VM, and then test it using the SoapUI plugin for NetBeans.

Task 1 – Define POJO Class

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

The simplest service one could imagine using in the Auction project might be an `ItemManager`, which would allow its clients to create/find/manage the set of persistent `Item` descriptions used by the system.

When sellers want to run auctions on the system, they have to enter a description for the item to be auctioned off - this is where `Item` instances come into play: instead of just creating a new Item description for each new auction started, the system wants to allow users to look for existing Items that match the new item the seller would want to put up for auction, so as to reuse that Item for as many auctions as are interested in that same description. Among other things, this might make it easier for the system to offer the ability to find similar auctions...

The `ItemManager` service then would be responsible for creating new `Item` instances, and finding existing `Item` instances that match a new item. It could also remove instances that are no longer used – but that is a more dangerous service to offer, within the Auction system. For convenience, limit your API to accept primitive types (and Strings), and to return the same: for example, the function to add a new `Item` could return the new `Item`'s id, and the function to remove an `Item` could take the id of the `Item` to remove. You may want to wait to implement the operation to find an `Item` until a later lab – this function would return an `Item`, given its id.

1. Create a new Java Application project; call it `WebServices`. For convenience, you should place this new project in the `exercises` folder in your home directory – that way, it will be kept separate from the two folders, `examples` and `projects`, that contain all the code provided to you.

2. Associate the existing `AuctionApp` project to your newly created project, so you can use all the types provided to you without modification. If you

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Aula Matriz S.A.S.

do not currently have the `AuctionApp` project open, go ahead and open it, first – it will be easiest to associate other open projects to your newly-created project.

You will be writing new web services, so your `WebServices` project should depend on the two libraries for JAX-WS 2.2 and JAXB 2.2.

Since the `WebServices` project will depend on `AuctionApp`, and it in turn uses JPA 2.0, your project will need to specify that it depends on JPA 2.0, as well – even though you may never refer to anything there explicitly. In NetBeans, the JPA 2.0 runtime is represented by the standard library named `EclipseLink (JPA 2.0)`.

The same goes for the Derby (or Java DB) client library: since your project depends on `AuctionApp`, and it depends on `DerbyJDBC`, so will your project, too.

3. Create a new class `ItemManager`, in package `labs`. Define as its public API the functions discussed above.

4. Implement the business logic associated with each of the functions in the interface for `ItemManager`.

Task 2 – Incorporate JAX-WS Annotations

Once the `ItemManager` is implement and working properly, it is time to turn it into a web service provider. Incorporate JAX-WS annotations, to turn your `ItemManager` into a web service.

Since JAX-WS is incorporated in Java SE 6, there should be no need to modify the classpath for the `WebServices` project to add support for it. However, Java SE 6 actually has one of a couple of different versions of JAX-WS built in, depending on which update is installed. To play it safe, make sure your project depends on the JAX-WS 2.2 (and JAXB 2.2) libraries provided by NetBeans.

Task 3 – Customize Project Build Script

Once we have modified `ItemManager` so that it can be used as as web service implementation class, we need to generate all the additional artifacts that JAX-

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Aula Matriz S.A.S.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

WS requires in order to actually deploy the web service. This may require additional tasks in the project's `ant` script, `build.xml`.

The elements that need to be added to the NetBeans `build.xml` file are listed in Code 2.1. They can also be found in the `build.xml` file for the project named `Managers` in the `examples/jaxws` folder – it is the examples project that contains all the standalone JAX-WS examples used throughout the course.

```
<taskdef name="wsgen"
         classname="com.sun.tools.ws.ant.WsGen"/>
<target name="-post-init" depends="">
  <mkdir dir="generated"/>
  <mkdir dir="generated/xml"/>
  <mkdir dir="generated/src"/>
</target>
<target name="-post-compile" depends="-post-init">
  <wsgen
      sei="labs.ItemManager"
      destdir="build/classes"
      resourcedestdir="generated/xml"
      sourcedestdir="generated/src"
      keep="true"
      verbose="true"
      genwsdl="true"
      xendorsed="true"
      protocol="soap1.1">
    <classpath>
      <pathelement path="${build.classes.dir}"/>
      <pathelement
        location="${reference.AuctionApp.jar}"/>
    </classpath>
  </wsgen>
</target>
```

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Code 2.1: Elements needed in `build.xml` for “Code-First”

Once you have modified your project's `build.xml` file, build the project once again. After the build completes successfully, you should find two new directories within your project:

- `generated/src` –
  This directory will contain the source code for the Java artifacts generated for your `ItemManager` web service.

- `generated/xml` –

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Aula Matriz S.A.S.

This directory will contain the WSDL/XML artifacts generated for your `ItemManager` web service.

In addition to these, the compiled versions of all Java artifacts will be placed in the `build` folder, along with all your other compiled classes.

Although the compiled code for the generated Java classes is present in the `build/classes` directory, the IDE may refuse to accept that these classed exist, when compiling your own code. To solve this issue, you should add the folder where the JAX-WS tools will place generated source classes (in this example, `generated/src`) as a *source folder* to the IDE. To add source folders to a project, right-click on your project, then choose `Sources` in the tab on the left. Initially, your project will have a single source folder called `src`. Add a second source folder for `generated/src`.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Task 4 – Deploy Web Service to Standalone JVM

Deploy your `ItemManager` web service to a standalone JVM. The simplest way to do this may be:

1. Create a new class, call it `ItemManagerRunner`, to be the application that deploys your web service. This new class will have a `main()` function.
2. In `main()`, create an instance of your web service implementation class `ItemManager`, and deploy it as a web service.
3. Ensure that the JavaDB database engine is already running.
4. Run this application.

As a simple test that your web service actually deployed successfully, remember that you can ask your web service to supply you with its WSDL description.

Task 5 – Test Web Service Using SoapUI

More through testing may be appropriate, however. Since we have not yet discussed how to create web service client applications that consume web services, we cannot write our own test client from scratch.

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Aula Matriz S.A.S.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

There is a tool we can use, instead, called `SoapUI`. It is available as a plugin for IDEs, or as standalone application – which you will find installed on your workstation. This tool will read the WSDL description of a web service, then generate a simple test harness that will help you write SOAP requests to deliver to the web service, then show you the return SOAP structure that your service delivers back to the calling client.

A quick tutorial on how to use `SoapUI` to test a web service can be found in the `resources` folder in your home directory, or directly from the authors' web site: `http://www.soapui.org/gettingstarted/your_first_soapUI_project.html`

The URL for the web service you would provide the `SoapUI` application with is constructed from the one that you indicated when publishing your service in the previous step – remember that you can add the suffix `?WSDL` to the web service URL, in order to request the WSDL description for that same service. You might use something like:

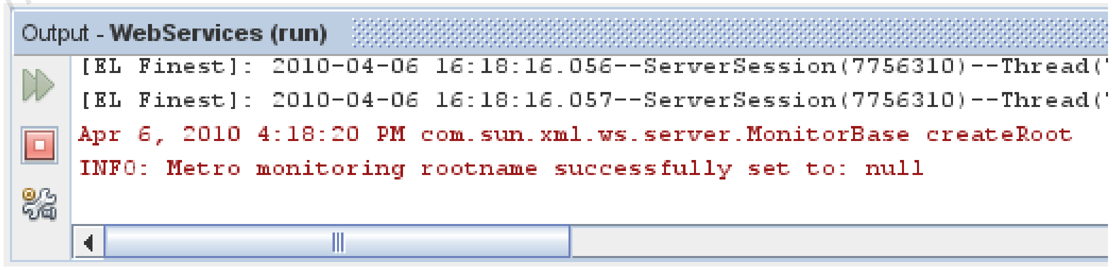
```
http://localhost:8081/ItemManagerService?WSDL
```

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Task 6 – Shut Down Web Service

When you started your web service application running, NetBeans displayed an output panel that contains the output produced by that application. That same panel includes the controls required to shut down, or restart, that application. Figure 2.3 illustrates that output panel, including those two controls:

Restart application   Stop application



```
[EL Finest]: 2010-04-06 16:18:16.056--ServerSession(7756310)--Thread('
[EL Finest]: 2010-04-06 16:18:16.057--ServerSession(7756310)--Thread('
Apr 6, 2010 4:18:20 PM com.sun.xml.ws.server.MonitorBase createRoot
INFO: Metro monitoring rootname successfully set to: null
```

Figure 2.1: Application Output for `WebServices` Project

In addition to that, NetBeans includes GUI elements that advise the user that there are applications running within NetBeans, and which allow the user to shut them down.

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Aula Matriz S.A.S.

Exercise 3 – Create, Deploy, and Test a JAX-WS Web Service Contract-First

In this exercise, we're going to create, deploy, and test a simple web service for the Auction application. We will implement this web service using a "contract-first" approach, describing first the API that we intend for our web service to implement, using the WSDL vocabulary to create our description. Once the WSDL description is ready, we will then implement the Java class that will be the service provider. Once we have our web service ready, we will deploy it to a stand-alone Java VM, and test it using the SoapUI plugin for NetBeans.

Task 1 – Define Abstract Description for Web Service in WSDL

Another simple service that the Auction system will need is a `UserManager` service, which allows people to register with the system, maintaining persistent objects for each user that has registered. The activities that the application would need to support, as far as user management is concerned, include:

- Add new users to the application. This might involve specifying the user's login id, full name, and email address. It could also include a password, to authenticate the user.
- Update information about an existing user, such as their full name, or their email address.
- List all users, or just the users that match some criteria.
- Remove users from the application.

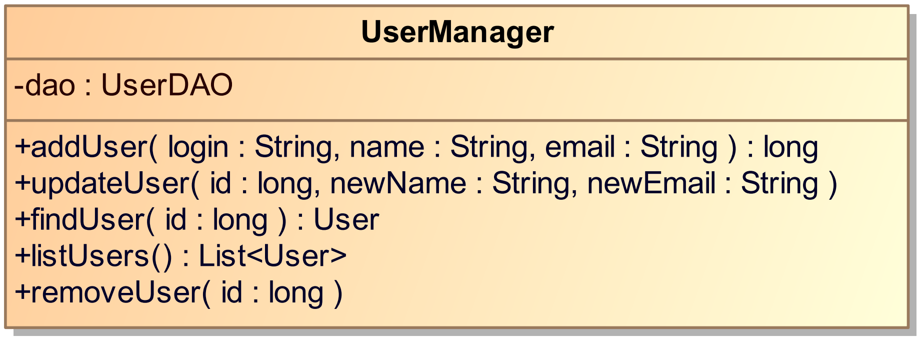

UserManager
-dao : UserDAO
+addUser(login : String, name : String, email : String) : long
+updateUser(id : long, newName : String, newEmail : String)
+findUser(id : long) : User
+listUsers() : List<User>
+removeUser(id : long)

Figure 2.2: Possible UML Diagram for `UserManager` Class

Figure 2.5 shows what a UML diagram for `UserManager` might look like.

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Aula Matriz S.A.S.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

1. Identify the basic functions that such a `UserManager` would need to support, including their mode of interaction, and parameters and return types.

2. Write the abstract WSDL description for this web service. You could name the file `UserManager.wsdl`, in package `labs`. This abstract description is the one captured in the `portType` and `message` elements of the WSDL file, along with the proper XML schema types, if appropriate.

   Name the port type `UserManager`, to be consistent with the UML diagram shown above.

   You can split the description of the service into several files (XML schema, logical WSDL description, WSDL bindings) as shown during lecture. It is also the way the description for the `PassengerManager` service in project `Managers` is organized. Alternatively, you can just write a single description file.

   If you chose to separate logical from concrete description, at this point you should have three files:

   - `UserManager.xsd` –
     An XML Schema file that contains the definitions for all the information that will be carries by any messages between client and web service provider.

   - `UserManager.wsdl` –
     The logical description of the web service, in terms of the set of operations that this service will support. This description depends on the XML Schema types defined above.

   - `UserManagerSvc.wsdl` –
     The concrete description of the service, which includes the information necessary to actually communicate with the service. This description depends on the logical description defined above.

   If you choose to capture all the elements that describe the `UserManager` service in a single XML file, name it `UserManagerSvc.wsdl` – just so that your exercise remain consistent with these instructions.

You can limit your API to functions that will only require or produce simple data. We will revisit this contract in a later chapter, once we learn how to write more complex WSDL files and their corresponding XML Schema types.

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Aula Matriz S.A.S.

In principle, you should write the WSDL description for this new `UserManager` service from scratch. However, writing WSDL descriptions can be quite cumbersome, until one gains enough experience. You could start with a template of the description you will need to write, instead – grab a copy of the WSDL description for the `PassengerManager` service from the `Managers` example project, and use it as a starting point for your own.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Task 2 – Incorporate Concrete Description for Web Service into WSDL

Complete the WSDL description for our new `UserManager` service by incorporating concrete details to the WSDL file. Remember that this concrete information is captured in the `binding` and `service` elements in the WSDL file.

The kinds of information included as "concrete details" in a WSDL file include: the WSDL style (document literal vs. RPC literal), the address where web service will be found, and any `SOAPAction` header.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Task 3 – Customize Project Build Script

In the "contract-first" approach, there are more artifacts to be generated from the WSDL definition, and the actual generation has to be requested explicitly – in order to write any code in Java that refers to the service that was initially described contract-first, the developer will need a Java interface for that service, and the only way to get it is by running the proper code generator.

1. Modify your build script to include steps require to trigger the generation of artifacts before compilation of the source code for your project.

   The elements that need to be added to the NetBeans `build.xml` file are listed in Code 2.2. They can also be found in the `build.xml` file for the project named `Managers` in the `examples/jaxws` folder – it is the examples project that contains all the standalone JAX-WS examples used throughout the course.

2. Build your `WebService`, to trigger the generation of all Java artifacts associated with the `UserManager` service.

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Aula Matriz S.A.S.

```
<taskdef name="wsimport"
         classname="com.sun.tools.ws.ant.WsImport"/>
<target name="-post-init" depends="">
  <mkdir dir="generated"/>
  <mkdir dir="generated/src"/>
</target>
<target name="-pre-compile" depends="-post-init">
  <wsimport
    wsdl="${basedir}/${src.dir}/labs/UserManagerSvc.wsdl"
    destdir="build/classes"
    sourcedestdir="generated/src"
    keep="true"
    verbose="true"
    extension="true"
    xendorsed="true"
    package="labs.generated">
  </wsimport>
</target>
```

Code 2.2: Elements needed in `build.xml` for "Contract-First"

Task 4 – Implement Web Service Based on Existing WSDL Description

The code generation step produces a Java interface (which will be named after the port type that you specified in your WSDL description) that describes what the API will be that the web service will require from anyone that is used as the implementation of that web service. Now that we have this information available, you can go ahead and implement the web service implementation class.

Task 5 – Deploy Web Service to Standalone JVM

Once the web service is implemented in Java, there's really very little difference between a "code-first" web service and a "contract-first" one. Go ahead and add the logic necessary to deploy your `UserManager` as a web service – you can use the same strategy that you used for the "code-first" implementation of the `ItemManager` web service that you built before:

1. Create a new class, call it `UserManagerRunner`, to be the application that deploys your web service. This new class will have a `main()` function.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Aula Matriz S.A.S.

2. In `main()`, create an instance of your web service implementation class `UserManagerImpl`, and deploy it as a web service.

3. Ensure that the JavaDB database engine is already running.

4. Run this application.

Task 6 – Test Web Service Using SoapUI Plugin

1. Use the SoapUI plugin to generate a test harness for your "contract-first" web service. Check that the web service behaves as advertised.

2. Once you have tested your web services, shut down the web service provider application.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Aula Matriz S.A.S.

Exercise 4 – Create and Run a Web Service Client

In this exercise, we're going to create and test a simple web service client to the `ItemManager` web service create in Exercise 2.

Task 1 – Define Client Class

1. Create a new project in NetBeans. Call it `WSClients`. Switch to it. This project won't require any dependencies on existing projects – given a WSDL description of the service that the client will use, JAX-WS will generate all the necessary Java artifacts.

   It is convenient to create web service clients in a different project than the one use to create the web service provider for a couple of reasons:

   - The *Separation of Concerns* principle suggests that web service providers and their matching web service clients ought to be maintained separately.
   - The supporting artifacts required by the web service provider and the web service client, or the way in which those artifacts are created, may not be the same. Generating both in the same project could be confusing.

2. Configure the project to support web service calls to the `ItemManager` web service.

   Instead of changing the `build.xml` file to run `wsimport` to generate the necessary Java artifacts explicitly, you will use a wizard in the NetBeans to prepare this project to support web service client logic:

   - Ensure that the `ItemManager` service is up and running.
   - Ask the IDE to configure a web service client reference within the `WSClients` project. When the wizard asks for the WSDL for the target service, enter the URL that can be used to query the service for its own WSDL description:

     ```
     http://localhost:8081/itemManager?WSDL
     ```

     Specify `labs.generated` for the target package for all artifacts.

   This step does not actually generate any client logic – it just configures the project so as to support the generation of client logic on demand. It

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Aula Matriz S.A.S.

also modifies the project to generate the web service artifacts needed by the client automatically.

Once again, NetBeans may refuse to recognize that the artifacts generated by JAX-WS exist. The easiest way to fix this is to add the directory where the generated code is to your project's set of source folders. The folder where these artifacts will appear will be `build/generated-sources/jaxws`.

---

3. Create a new main class, `clients.ItemManagerClient`.
4. Write logic in your main class to invoke an operation on the web service created in Exercise 2, the `ItemManager` web service.

Task 2 – Execute Client

1. Run the `ItemManager` web service in `Lab02`, if it is not currently running.
2. Run the `ItemManagerClient` application.
3. Once you have tested your web service, shut down the web service provider application.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Aula Matriz S.A.S.

Lab Summary

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

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Aula Matriz S.A.S.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Aula Matriz S.A.S.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Lab 3

SOAP and WSDL

Objectives

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

- Write WSDL descriptions of web services, using the different styles of SOAP messages that are available.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Aula Matriz S.A.S.

Exercise 1 – Understanding SOAP and WSDL

1. Identify the most popular model and style of SOAP messages from the following list. (Choose one).

   (a) Document literal

   (b) Document encoded

   (c) RPC literal

   (d) RPC encoded

2. State whether the following statements are true or false:

   (a) A WSDL descriptor for a web service is required only for a SOAP-based web service.

   (b) A WSDL file is embedded within a SOAP message.

   (c) WSDL was designed to serve as a standard mechanism for defining the functionality exposed by a web service.

   (d) A web service client can retrieve a service descriptor only from the registry.

3. Identify the primary elements of WSDL from the following list. (Choose one).

   (a) Definitions, types, message, portType, binding, service.

   (b) Types, portTypes, part, input, and output.

   (c) Definitions, part, port, service, and binding.

   (d) Definitions, portType, operations, service, and binding.

4. Identify one or more ways by which the WSDL file of a specific web service is located:

   (a) `http://eshop.com/CustomerServices/POService?WSDL`

   (b) `http://eshop.com/CustomerServices/POService=WSDL`

   (c) `http://eshop.com/CustomerServices/POService/WSDL/OrderService.WSDL`

   (d) `http://eshop.com/CustomerServices/POService?wsdl=Orderservice.wsdl`

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Aula Matriz S.A.S.

5. State whether the following statements are true or false:

   (a) A WSDL descriptor has an abstract model and a concrete model to describe messages and operations.

   (b) There are only two types of messaging mechanism; one way and Request/Response.

   (c) The binding element specifies the data format and the protocol used in the web service. The standard binding extensions are HTTP, SOAP, and MIME.

   (d) A WSDL file alone cannot be used to generate a web service.

6. Given:

```
<soap:envelope>
  <soap:body>
    <add>
      <x xsi:type="xsd:int">5</x>
      <y xsi:type="xsd:int">5</y>
    </add>
  </soap:body>
</soap:envelope>
```

   What is the style/encoding for this SOAP message? (Choose one).

   (a) RPC/encoded

   (b) RPC/literal

   (c) Document/literal

   (d) Document/literal wrapped

7. Which style specifies that each message consist of a single XML node that corresponds to the operation being invoked, with values encoded as simple text? (Choose one).

   (a) RPC/encoded

   (b) RPC/literal

   (c) Document/literal

   (d) Document/literal wrapped

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Aula Matriz S.A.S.

Exercise 2 – Design a Web Service "Contract-First" Using the RPC/Literal Style

In Lab 2, you implemented the `UserManager` web service "contract-first", by starting with its WSDL description. At the time, you had only seen the Document/literal "wrapped" style of WSDL description – so that is the kind of description that you wrote. In this module, you learned about the different styles of WSDL descriptions that are available.

In this exercise, you will implement that same web service "contract-first", too – but using the RPC/literal style of WSDL description. Once the WSDL description is ready, you will then implement the Java class that will be the service provider. and then deploy it to a stand-alone Java VM. You can test this service with the SoapUI, too. In the next exercise, you will write a Java client for this service.

Task 1 – Define Description for Web Service in WSDL

1. Create a new project, call it `RPCWebServices`. Switch to it.

   Since you will be running web service implementation classes for the Auction domain out of this project, it will need to have the same dependencies you had specified for the `WebServices` project: the `AuctionApp` project, and the `EclipseLink (JPA 2.0)`, `JAX-WS 2.2`, `JAXB 2.2`, and the Derby (or Java DB) libraries.

2. Create a Java package called `labs`.

3. Write a second complete WSDL description for the `UserManager` web service, using the RPC/literal style. Since you are working on a new project, you could keep same file name conventions that you used back in Lab 2: name the file `UserManagerSvc.wsdl`, and place it in the `labs` package (folder) you just created.

   You can limit your API to the same functions originally described in Lab 2, Exercise 2.

Task 2 – Customize Project Build Script

In the "contract-first" approach, there are artifacts to be generated from the WSDL definition, and the actual generation has to be presented explicitly – in

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Aula Matriz S.A.S.

order to write any code in Java that refers to the service that was initially described by its contract, the developer will need an Java interface for that service. The only way to get that interface is by running the code generator explicitly.

Modify your build script to include the steps require to trigger the generation of artifacts before compilation of the source code for your project – just as you did in Lab 2.

Also, remember to add the `generated/src` folder, where the artifacts generated will be placed, as another source folder within the `RPCWebServices` project; otherwise, NetBeans won't find those artifacts when compiling your own code.

Task 3 – Implement Web Service based on Existing WSDL Description

The code generation step produces a Java interface that describes what the API will be that the web service will require from anyone that is used as the implementation of that web service. Now that you have this information available, go ahead and implement the web service class.

Task 4 – Deploy Web Service to Standalone JVM

Once the web service is implemented in Java, there's really very little difference between a "code-first" web service and a "contract-first" one. Go ahead and add the logic necessary to deploy your `UserManager` as a web service – you can use the same strategy that you used for the "code-first" implementation of the web service that you built in Lab 2, or the one that you used for the "contract-first" implementation, also in Lab 2.

To avoid confusion, and to be able to run this version of the `UserManager` service along with the earlier one, specify a different URL when you `publish` the web service in your server application.

Task 5 – Test Web Service Using SoapUI Plugin

Use the SoapUI plugin to generate a test harness for your "contract-first" web service. Check that the web service behaves as advertised.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Aula Matriz S.A.S.

Exercise 3 – Clients for "Contract-First" Web Services

In this exercise, you're going to create and test a simple web service client to the new "contract-first" `UserManager` web service defined above.

Task 1 – Define a Client Class

1. Create a new Java Application project, call it `RPCClients`. Switch to it. Remember that client projects need not specify any application-level dependencies – the Java artifacts they need will be generated from the WSDL description to the service they will interact with.
2. Configure this new project to support web service client interactions with the new `UserManager` service.
3. Explicitly build the `RPCClients` project.
4. Create a new main class `clients.UserManagerClient`.
5. Write logic in your main class to interact with the `UserManager` web service created in Exercise 2 above. The simplest way to do this may be to use the NetBeans wizard to generate call a web service operation within `main()`.

Task 2 – Test WSDL Clients

Run each of the two `UserManager` clients against their corresponding web service provider applications, to verify they both work:

1. Verify that the JavaDB database server is already running.
2. Run each of the two `UserManager` clients.
3. Once you have tested your web services, shut down the web service provider applications.

Task 3 – Examine Database – Optional

Interacting with your web services could result in the insertion or update of rows into the database tables that represent the persistent entities managed

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

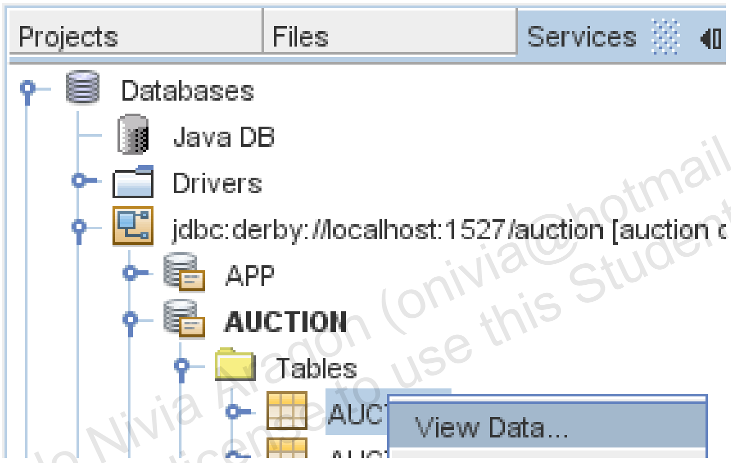
Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Aula Matriz S.A.S.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

by your web services. You could examine these database tables, to view the changes caused by your web service interactions:

1. Switch to the Services panel in NetBeans.
2. Right-click on the node that represents a connection to the `auction` database, and select `Connect` to establish that connection to the database engine.
3. Expand the connection node, then expand its `Tables` child node.
4. Right-click on the node that represents a table manipulated by one of your web services, then select `View Data`. Figure 3.1 shows what the pop-up dialog might look like.



Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Figure 3.1: View Data Popup Dialog

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Aula Matriz S.A.S.

Lab Summary

Compare the WSDL descriptions for the two variants of the `UserManager` service, the matching Java interfaces generated for each, and the corresponding web service client applications.

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

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Aula Matriz S.A.S.

Lab 4

JAX-WS and JavaEE

Objectives

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

- Deploy POJO web services to a web container.
- Secure POJO web services to a web container, delegating authentication to the container.
- Define a web service in terms of an Enterprise Java Bean.
- Deploy an EJB web service to an EJB container.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Aula Matriz S.A.S.

Exercise 1 – Understanding JAX-WS in Containers

1. Which are considered advantages of deploying POJO web services to a web container? (Choose all that apply).

   (a) HTTP session management built in.

   (b) Scalability and availability features built-in.

   (c) Declarative transaction management.

   (d) Pooling of web service provider instances built in.

2. Which mechanisms are available to define the set of roles that will be available for programmatic authorization within a web service provider class? (Choose all that apply).

   (a) Initialization properties in `web.xml` element corresponding to web service implementation class.

   (b) `@DeclareRoles` annotation on web service implementation class.

   (c) `@RolesAllowed` annotation on web service implementation class.

   (d) `security-constraint` element in `web.xml`.

   (e) `@DeclareRoles` annotation on web service client implementation class.

   (f) `security-role-mapping` elements in `sun-web.xml`.

3. What mechanisms are available for a JAX-WS web service deployed to a web container to obtain the identity of the caller? (Choose all that apply).

   (a) `ServletRequest` as a parameter to the web service method call.

   (b) `ServletRequest` as a resource injected directly in the web service implementation class.

   (c) `SecurityContext` as a resource injected directly in the web service implementation class.

   (d) `ServletRequest` obtained from a `MessageContext`.

   (e) `WebServiceContext` as a resource injected directly in the web service implementation class.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Aula Matriz S.A.S.

4. Choose the option that best describes how most of the information about a request that is made available to the web service implementation class is provided. (Choose one).

   (a) Directly via dependency injection.

   (b) As properties provided by `WebServiceContext` directly.

   (c) As properties provided by `MessageContext` directly.

   (d) As parameters to the web service method call.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Aula Matriz S.A.S.

Exercise 2 – Deploy POJO Web Services to a Web Container

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

In Lab 2, you built two simple web services, `ItemManager` (implemented code-first) and `UserManager` (implemented contract-first). They were both designed to be simple POJO web services, and they were both deployed standalone on a JVM.

In this exercise you will deploy them into a web application, so as to leverage the functionality provided by the web container: initially, you will gain the ability to use the automated test harness provided by GlassFish. Later, you will be able to authenticate clients of the web service by delegating the authentication logic to the web container.

POJO web services can be deployed either standalone, using the HTTP server built into Java SE 6, or to any web container. The implementation of the POJO web service itself does not have to change, due to the environment it will be deployed into – any differences at that level are handled by the JAX-WS runtime.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Task 1 – Copy Existing Web Services to Web Application

1. To avoid confusion, create a new web application project (give it the name `POJOsInContainer`) to deploy the two web services into. As before, this new project will need to depend on the `AuctionApp` project and Derby library. (You need not specify either of the JAX-WS 2.2 or JAXB 2.2 libraries, as these are built into GlassFish v3).

   Make sure that you choose to create the project as a `Java EE 6` project.

2. Add to this project two server resource configuration entries:

   (a) Configure a database connection pool, so that this application (and, in particular, the persistence layer that your services rely on) can access the database efficiently. You can name this pool `auctionDbPool`. Use the NetBeans connection to the `auction` database that you configured in in the setup lab, so that NetBeans can configure the connection pool based on that same configuration.

   (b) Configure a JDBC resource to let the application access the database through the `auctionDbPool` via JNDI lookup. The name of this JDBC resource must be `jdbc/auction`, as that is the resource name em-

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Aula Matriz S.A.S.

bedded in the JPA configuration file for the `AuctionEJBs` project that your project depends on.

3. Copy the code for the two services `ItemManager` and `UserManager` from the `WebServices` project you created in Lab 2 into this new project.

   To avoid having to make changes to the `build.xml` script in this new project, you could use NetBeans wizards to help you introduce these web services to this web application:

   (a) Use the NetBeans "create new web service" wizard to create a new empty `ItemManager` web service "code-first" (from scratch). Place it in the `labs` package.

   (b) Once the empty web service implementation class is present, copy the implementation of that class from the `ItemManager` class in the `WebServices` project.

   (c) Copy the WSDL description for the `UserManager` web service from the `WebServices` project into the `labs` folder of `POJOsInContainer`.

   (d) Use the NetBeans "create new web service" wizard again, to create a new web service from a given WSDL description. Have the wizard look at the WSDL description for `UserManager` in this project.

   (e) Copy the definitions of the members of the `UserManagerImpl` class from the `WebServices` project to this one.

4. No explicit configuration changes to the web application (that is, changes to the `web.xml` file) are needed – JAX-WS will configure URLs to access the web services within the application by default, based on their class name. The URLs to access services will have the form:

   ```
   http://host:port/context/serviceName
   ```

   where *`serviceName`* by default will be the class name for the POJO web service, with the suffix `Service`. However, it is possible to control the service name component of that URL: the `serviceName` attribute of the `@WebService` annotation allows the developer to specify the name clients will use to contact the web service in question. Change the definition of the `ItemManager` web service so that clients can contact it at the URL:

   ```
   http://localhost:8080/POJOsInContainer/ItemManagerWS
   ```

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Oscar Hernando Nivia Aragon (olivia@hotmail.com) has a non-transferable license to use this Student Guide

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Aula Matriz S.A.S.

Task 2 – Deploy and Test Services in Web Application

Deploying the `POJOsInContainer` web application to a web container will automatically deploy the web services contained within it.

A benefit of deploying JAX-WS services to a web container is that an automated test harness for those web services is provided for free. The `ItemManager` web service is available at the URL:

```
http://localhost:8080/POJOsInContainer/ItemManagerWS
```

To see this test harness in action:

1. Deploy the `POJOsInContainer` web application.
2. Open up a browser, and visit the URL:
   `http://localhost:8080/POJOsInContainer/ItemManagerWS?Tester`

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Aula Matriz S.A.S.

Exercise 3 – Secure POJO Web Services in Web Container

Earlier, you deployed the two web services `ItemManager` and `UserManager` to a web container, as part of a web application. One of the advantages of doing this is that it becomes possible to secure those web services, delegating authentication to the web container. In this exercise, you will secure access to the `UserManager`, so that only administrators can create new users.

Task 1 – Define Security Constraints on Web Service

You have to capture, as part of the description of the service, that there are two kinds of users of the `UserManager` web service, "administrators" and everyone else, and that only administrators are allowed to add new users to the application:

1. For convenience, let's define two *roles*, or groups of users, called `user` and `administrator`. The set of roles expected by a JAX-WS web service can be captured by annotating the web service implementation class with a `@DeclareRoles` annotation, listing those roles by name.

2. A `@RolesAllowed` annotation can be specified at class level, to specify a default requirements that the caller belong to certain roles to call any operation on the web service implemented by that class. It can also be specified at method level, to restrict access to a particular operation in that class. Capture the constraints that callers in the `user` role can call any of the operations in either `ItemManager` or `UserManager`, but only callers in the `administrator` role can call the `addUser` operation in `UserManager`.

You could also log the name of the person adding a new user to the container's log file, just to show that authenticated user information is provided to the web service.

Task 2 – Define Security Constraints on Web Application

Capturing authentication constraints at the level of the web service implementation classes isn't enough. JAX-WS will delegate authentication to the web container – but the web container requires additional configuration information, in order to perform authentication:

1. Create an empty `web.xml` configuration file, if it doesn't exist, yet.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Oscar Hernando Nivia Aragon (ornivia@hotmail.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Aula Matriz S.A.S.

2. A `login-config` element has to be added to the web application's configuration file, `web.xml`. This element is used to configure the authentication mechanism that the web container will use. Set it to `BASIC`, which expects clients to provide userid and password information in an HTTP header. Set the realm name to `file` – which is the simplest realm supported by GlassFish.

3. Add `security-role` elements for the two roles needed in this exercise, `user` and `administrator`.

4. A `security-constraint` element has to be added to `web.xml`, too. Label it `AuctionServices`.

   This element indicates which URLs are to require user authentication, for a given HTTP method, and which roles a caller has to belong to, in order to be allowed access to the resource. Set the HTTP method to `POST` only – clients will need to request the WSDL description of the service from it, and this is done via a `GET` request.

5. The set of users expected by the application, along with the roles that each user will belong to, must be specified, too - but this is not part of the product-independent web application configuration.

   In the case of GlassFish in particular, the product-specific configuration file is `sun-web.xml`. Add `security-role-mapping` elements to this file, for each of the users that will require authentication to use the application.

   The lab instructions assumes there will be two users, with userids `tracy` and `kelly`; their passwords will be set to `password` in a later step. `tracy` is assumed to be an `administrator`, while `kelly` is a regular `user`.

The set of roles that can be specified in the web application's `sun-web.xml` is *application-specific* – different web applications deployed to the same web container could each have different roles defined for each application.

For convenience, `sun-web.xml` can also map *system-wide* roles to application-specific ones – this would allow for the administration of a number of web applications, without requiring a complete of all authorized used separately for each individual application: the system-wide assignment of system roles to principals would determine their application-specific role assignments within each web application.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.
Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Aula Matriz S.A.S.

Task 3 – Configure Users in Authentication Realm

The actual authentication of principals is outside the scope of the specification of a web application – it is a responsibility of the web container. Therefore, configuration of the information associated with known principals (for example, their user names and passwords) must be done at the level of the web container.

In the case of GlassFish, the ability to define principals is available through the GlassFish *administration console*. This administration console is a web application deployed within GlassFish, available at the url:

```
http://localhost:4848
```

To access it, you may need the GlassFish administrator's login (which by default is `admin`) and password (by default `adminadmin`). You may have to start the server, if it is not yet running, before you can access its administration console.

You can also access the GlassFish administration console through NetBeans: right-click on the node for `GlassFish v3 Domain`, in the `Services` tab, as shown in Figure 4.6.

Once on the administration console's home page, the security configuration page can be found under `Configuration/Security` in the navigation bar on the left. Configuration of the default file realm (specified in that main security configuration page) can be found under `Configuration/Security/Realms/file`. You can click on the `Manage Users` button there, to add users and their credentials to the realm.

The lab instructions assumes there will be two users, with userids `tracy` and `kelly`. Both of them should have their passwords set to `password`.

Task 4 – Deploy Service in Web Application

Since the two web services are now packaged up as part of a web application, deploying that web application will automatically deploy the two web services along. Go ahead and deploy the `POJOsinContainer` web application.

Deploying this application should start the database server running automatically, if it was not yet running. Ensure that the database server is running, when the application is deployed.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Aula Matriz S.A.S.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

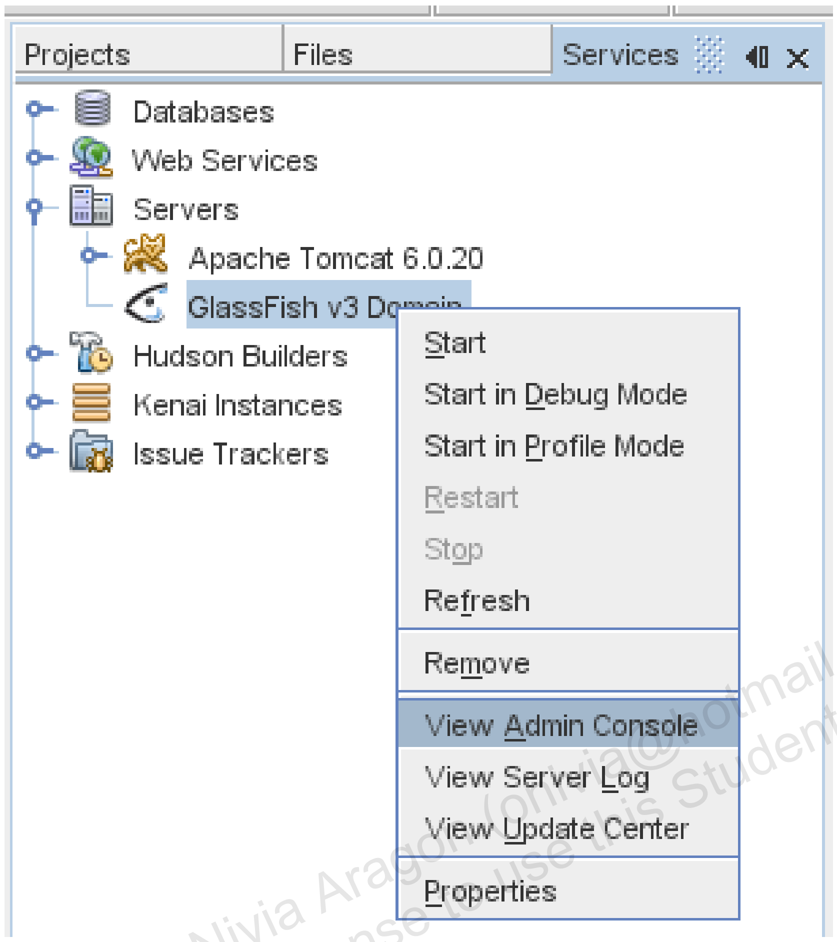


Figure 4.1: NetBeans Link To GlassFish Admin Console

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Normally, you could use the test harness provided in GlassFish to test web services – except that the test harness is not sophisticated enough to pass authentication information as part of the test request. To test the secured web services, you will have to write a JAX-WS client application that can authenticate itself when it calls out to the web services.

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Aula Matriz S.A.S.

Exercise 4 – Creating an Authenticating Client

In this exercise, you're going to create and test a simple authenticating web service to the secured `UserManager` web service defined above.

Task 1 – Define a Client Class

1. Create a new Java Application project, call it `SecuredClient`. Switch to it. Remember that client projects need not specify any application-level dependencies.

To work around a bug in JAX-WS, JAXB, and GlassFish, please add the following XML element to the `build.xml` script for your project:

```
<taskdef name="wsimport"
         classname="com.sun.tools.ws.ant.WsImport"/>
```

Place this line after the `import` element in that file.

2. Configure this new project to support web service client interactions with the new secured `UserManager` service.

The NetBeans wizard will ask you what package you would like to place the generated artifacts into. You **must** leave this blank! There seems to be a bug in the way JAX-WS and JAXB are handling XML namespaces – and the workaround is to let them default the artifacts' package name to match the namespace name.

3. Create a new main class `labs.UserManagerClient`.

4. Write logic in your main class to invoke the `addUser` operation on your secured web service. The simplest way to do this may be to use the NetBeans wizard to generate call a web service operation within `main()`.

   There is one catch, however. You will also need to provide authentication information for the client proxy to send as part of the web service request, so that the server can validate whether you should be allowed to execute that operation.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Aula Matriz S.A.S.

Task 2 – Test WSDL Clients

Run the `UserManager` client against its corresponding web service provider application, to verify it works.

Task 3 – Shut Down Web Services

Undeploy the web service provider application(s) you deployed in this exercise, and shut down the application server.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Aula Matriz S.A.S.

Exercise 5 – Define and Deploy EJB-based Web Services

The web services you have built so far use data access objects (DAOs) to manipulate persistent domain objects. If the task they need to perform becomes complex enough, the web services themselves may need to manage the transactions used to perform the work required by each operation – and this could be complex logic to write and maintain.

In this exercise, you will modify your web services so they are implemented as Enterprise java Beans, rather than POJOs. This will allow you to delegate the implementation of the transaction management that would be required to the EJB container – though you will still to specify (declaratively) what the transaction semantics required by your operations will be.

Task 1 – Copy Existing Web Services to Enterprise Application

As a new feature in JavaEE6, an enterprise application can be packaged up and deployed as a simple web application – an EJB container, and many of the services that enterprise applications rely on, are available alongside the JavaEE6 web container. In this lab, you will take advantage of this new feature.

1. Copy your existing `POJOsInContainer` web application project into a new project, called `EJBWebServices`. This will avoid confusion between the new EJB-based services you are about to write, and the services deployed earlier as POJO web services in a web container.

   However, you are going to modify your web services so that they are implemented as Enterprise JavaBeans. At the same time, the DAOs that the web service providers rely on will also become EJBs – and so you will benefit from the ability to manage transactions contexts transparently that is built into the EJB framework.

   You will achieve the switch from POJO DAOs to EJB DAOs by changing your project dependencies: instead of depending on the `AuctionApp` project, as you have been doing so far, you will now make this new project depend on the `AuctionEJBs` project (which is inside the same folder as the original `AuctionApp`).

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Oscar Hernando Niva Aragon (oniva@hotmail.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Aula Matriz S.A.S.

Until you modify your web service implementation classes, as described in the next step, NetBeans will report problems with those classes, after you update your project's dependencies. This is due to two small differences:

- The APIs are a little different, the POJO DAOs and the EJB DAOs.
- The package where the DAOs are placed is different.

So the errors are to be expected, until you fix your code in the next step.

Task 2 – Rewrite Existing Web Services as EJBs

Modifying the web service implementation classes so that they are implemented as EJBs is straightforward: since the two specifications (JAX-WS and EJB) can be said to be orthogonal to (that is, independent of) each other, you only need to add the specification that the class is to be an (stateless) EJB to the existing code – and fix the problems introduced by the change in dependent project.

The one important difference between POJOs and EJBs to keep in mind is that it is the responsibility of the application to instantiate POJOs, while it is the responsibility of the container to instantiate EJBs: you must modify your web service implementation classes so that, where your POJO web service implementation class used to instantiate a POJO DAO explicitly, you will now ask the container to inject an EJB DAO into your EJB-based web service implementation class.

Task 3 – Deploy EJb-Based Services in Web Application

Since the two EJB-based web services are also packaged up as part of a web application, deploying that web application will automatically deploy the two web services along. Go ahead and deploy the `EJBWebServices` web application.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Aula Matriz S.A.S.

Exercise 6 – Creating a Client for an EJB-Based Web Service

Task 1 – Create and Run a Client Application

Since the EJB-based web services you just created were based on the access-restricted POJO web services from an earlier lab, they too require authentication data from their clients: an authenticating client will be required, to test the EJB-based web services:

1. The simplest way to build an authenticating client for the new EJB-based web services is to copy the authenticating client you build in Exercise 4. However, to avoid confusion (and perhaps interference between clients), you should copy the complete `SecureClient` project into a new project, call it `EJBSecuredClient`.

2. Remove the reference to the `UserManagerSvc` currently configured in the `EJBSecuredClient` project. This is a reference to the web service in the `POJOsInContainer` service, which is not your intended target for this new client.

3. Add a web service client reference to the EJB-based `UserManagerSvc` in the `EJBWebServices` project. If you followed the naming suggestions before, the URL to this service should be:

   ```
   http://localhost:8080/EJBWebServices/UserManagerSvc
   ```

4. The existing `UserManagerClient` application should work with no changes, as a client for the new web service. Edit that class, if necessary, then execute it to test your EJB-based web service.

Task 2 – Shut Down Web Services

Undeploy the web service provider application(s) you deployed in this exercise, and shut down the application server.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Oscar Hernando Nivia Aragon (oriyia@hotmail.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Aula Matriz S.A.S.

Lab Summary

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

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Aula Matriz S.A.S.

Lab 5

Implementing More Complex Services Using JAX-WS

Objectives

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

- Apply JAXB to pass complex objects to and from a web service.
- Understand how to map Java exceptions from a web service provider to SOAP faults.
- Inject attributes into JAX-WS web service endpoints.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Aula Matriz S.A.S.

Exercise 1 – Understanding JAX-WS

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Given:

```
<message name="sayHello">
  <part name="parameters" element="tns:sayHello"/>
</message>
<message name="sayHelloResponse">
  <part name="parameters" element="tns:sayHelloResponse"/>
</message>
<message name="UDException">
  <part name="fault" element="tns:UDException"/>
</message>
<portType name="UserDefinedExceptionWS">
  <operation name="sayHello">
    <input message="tns:sayHello"/>
    <output message="tns:sayHelloResponse"/>
    <fault name="UDException" message="tns:UDException"/>
  </operation>
</portType>
```

1. Which is a valid endpoint signature for this WSDL fragment? (Choose one)

   (a) `public void sayHello() throws UDException`

   (b) `public String sayHello(String name) throws UDException`

   (c) `public String sayHello() throws UDException`

   (d) `public void sayHello(String name) throws UDException`

2. When do service exceptions occur? (Choose one)

   (a) When a web service call does not result in a fault.

   (b) When a web service call results in the service returning a fault.

   (c) Whenever a web service call occurs.

   (d) Whenever a web service call does not occur

Aula Matriz S.A.S.

3. Which parameter(s) is/are required to configure the constructor for `javax.xml.ws.soap.SOAPFaultException`?

   (a) fault

   (b) fault, actor

   (c) fault, actor, code

   (d) fault, actor, code, node

4. Which of the following technologies uses `SOAPFaultException` to wrap and manage a SOAP-specific representation of faults?

   (a) DOM

   (b) SAX

   (c) SAAJ

   (d) XML

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Aula Matriz S.A.S.

Exercise 2 – Pass Complex Types

The services you have implemented so far, `ItemManager` and `UserManager`, have been limited by the fact that you had only seen how to pass simple types. In this Module, you learned how to pass complex types, so you can make these two services more complete.

You should continue to work on the `EJBWebServices` project (where you have written EJB-based web services): extending the EJB-based web services is more straightforward, since your web services can delegate transaction management and security access control to the EJB infrastructure.

Task 1 – Complete the API of the ItemManager Service

1. Complete the APIs of the `ItemManager` web services, to include operations that require complex types as input or output parameters:

   (a) Add a `findItem` operation to your `ItemManager`, which accepts an id as a parameter, and returns the `Item` with that id.

   (b) Add a `updateItem` operation to your `ItemManager`, which accepts an `Item` as a parameter, to update the persistent version of that item.

2. Deploy the updated web service.

Task 2 – Update ItemManager Web Service Client

In earlier labs, you wrote standalone Java client applications to test the web services you built then – and in particular, you wrote a standalone client application to test the `ItemManager` web service. In this lab, you have updated that service to add new functionality. To test it, you will extend that earlier client to invoke the new operations you just added to `ItemManager`:

1. NetBeans projects which include web service client logic have a reference to the target web service as part of the configuration of those projects. This reference is used to generate the client-side Java artifacts that will be needed to call on that service – but, if the web service changes, the reference to the web service in the client project must be updated.

   Refresh that web service client reference, to update the Java artifacts available to the client application. You may also have to rebuild the project

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Aula Matriz S.A.S.

explicitly, to ensure that those Java artifacts are regenerated and recompiled.

2. Add logic to the client application to invoke the new operations available.

   Alternatively, you could copy the existing client application, to create a new application to invoke the new operations.

Task 3 – Complete the API of the UserManager Service

1. Complete the WSDL description of the `UserManager` web services, to include operations that require complex types as input or output parameters:

   (a) Add a `findUser` operation to your `UserManager`, which accepts an id as a parameter, and returns the `User` with that id.

   (b) Add a `updateUser` operation to your `UserManager`, which accepts an `User` as a parameter, to update the persistent version of that user.

2. Refresh the Java artifacts generated on the server side from your WSDL description. This will add new methods in the web service endpoint interface corresponding to the new operations you just added to the WSDL description of the web service.

   NetBeans caches a copy of the WSDL description of a "contract-first" web service – so it is probably a good idea to allow the wizard to replace that cached copy.

3. Add implementations of the new methods to the web service implementation class that you wrote, to actually provide the services included in the new WSDL description for the `UserManager` web service.

4. Deploy the updated web service.

Task 4 – Update UserManager Web Service Client

Earlier, you wrote a standalone client application to test the `UserManager` web service. In this lab, you have updated that service to add new functionality. To test it, you will extend that earlier client to invoke the new operations you just added to the service:

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Aula Matriz S.A.S.

1. Refresh the web service reference to the `UserManager` service in your client project, to update the Java artifacts available to the client application in that project.
2. Update the client application to invoke the new operations available.
3. Run the updated client application, to test that your new web services are operating correctly.

Task 5 – Shut Down Web Services

Undeploy the web service provider application(s) you deployed in this exercise, and shut down the application server.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Aula Matriz S.A.S.

Exercise 3 – Building More Complex Web Services Using JAX-WS

The web services you are implementing are there to support an Auction application. In this exercise, you will write some of the more higher-level services associated with such an application, using the code-first approach.

You will implement an `AuctionManager` web service. Some of the Auction operations require the `User` who is making the request (the seller, when creating an auction, or the bidder, when placing bids). You could implement the `AuctionManager` web service in your original `WebServices` project as another standalone web service – but your service could not verify the identity of the caller. It would be more realistic to build this new web service on either the `POJOsInContainer` or the `EJBWebServices` projects, so as to benefit from the ability to authenticate users offered to POJO web services, or to EJB-based web services. You will continue to work in the `EJBWebServices` project.

Task 1 – Design and Implement a AuctionManager Web Service

An Auction application must support, at a minimum, the ability for users to create and look up auctions, and place bids on existing auctions. Figure 5.1 could be a UML diagram for the operations that the `AuctionManager` would offer.

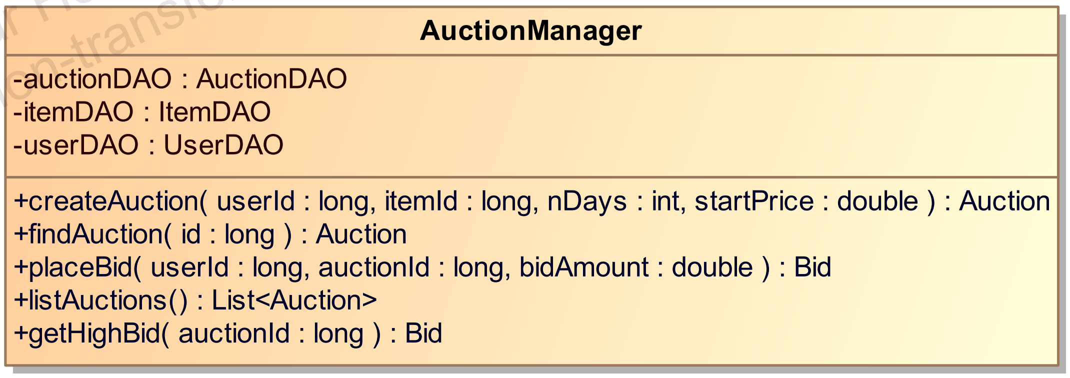

AuctionManager
-auctionDAO : AuctionDAO
-itemDAO : ItemDAO
-userDAO : UserDAO
+createAuction(userId : long, itemId : long, nDays : int, startPrice : double) : Auction
+findAuction(id : long) : Auction
+placeBid(userId : long, auctionId : long, bidAmount : double) : Bid
+listAuctions() : List<Auction>
+getHighBid(auctionId : long) : Bid

Figure 5.1: Possible UML Diagram for AuctionManager Web Service

1. Define a `AuctionManager` Web Service as the class `labs.AuctionManager` in your web service project.
2. Include the operations listed in Figure 5.1 above.

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Oscar Hernando Nivia Aragon (oh...ia@hotmail.com) has a non-transferable license to use this Student Guide.

Aula Matriz S.A.S.

3. Add the ability for the new web service to authenticate users:

   (a) Declare that all may call any operations on `AuctionManager`, using annotations on the Java class.

   (b) Require that the container obtain identity information for all callers of the `AuctionManager` service.

4. Deploy the updated web service project.

Task 2 – Create an AuctionManager Web Service Client

Since the `AuctionManager` web service is another authenticating web service, you will need to create another Java client application to interact with it:

1. Create a new client application to test your new `AuctionManager` web service in the `EJBSecuredClient` project.

2. Run your new client application, to test the `AuctionManager` web service.

Task 3 – Shut Down Web Services

Undeploy the web service provider application(s) you deployed in this exercise, and shut down the application server.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Aula Matriz S.A.S.

Lab Summary

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

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Aula Matriz S.A.S.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Aula Matriz S.A.S.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Lab 6

JAX-WS Web Service Clients

Objectives

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

- Understand how to create web service clients using JAX-WS.
- Understand how to create web service clients using JAX-WS that support asynchronous pull-based interactions.

Oscar Hernando Nivia Aragon (olivia@hotmail.com) has a non-transferable license to use this Student Guide.

Aula Matriz S.A.S.

Exercise 1 – Understanding How to Create Web Service Clients

1. Which is the proper annotation to inject a `Service` instance into a JavaEE web service client? (Choose one).

   (a) `@WebService`

   (b) `@WebServiceRef`

   (c) `@Resource`

   (d) `@EJB`

2. Which is the proper way to enable schema validation on a JAX-WS client? (Choose one)

   (a) Add a binding declaration to the WSDL description of the service.

   (b) Add the `@SchemaValidation` annotation to the web service client implementation class.

   (c) Create an instance of the proper `WebServiceFeature` subclass, then pass it to the `getPort()` call on a `Service`.

   (d) There is nothing to do: schema validation is enabled by default on JAX-WS web service clients.

3. Which is the proper way to enable support for asynchronous interactions in a JAX-WS web service port proxy? (Choose one).

   (a) Add the `@Asynchronous` annotation to the method to invoke asynchronously on the web service implementation class.

   (b) Add the `@Asynchronous` annotation to the field that will hold the reference to the JAX-WS port proxy on the client side.

   (c) Add a binding customization to the WSDL description of the web service, on the client side.

   (d) One cannot invoke asynchronous operations at the level of the JAX-WS proxy type; the only way to invoke an operation asynchronously in JAX-WS involves bypassing the proxy and using the dynamic JAX-WS `Dispatch` API.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Aula Matriz S.A.S.

Exercise 2 – Building an Asynchronous Web Service Client

Task 1 – Creating an Asynchronous Web Service Client

All your client applications so far have assumed that there was nothing else going on at the client, at the time that it issued a call for an operation on one of your web services – so it was reasonable for the client to simply issue the call, then wait for the answer to come back.

One could imagine a more sophisticated Auction client that presents its user with status information on multiple auctions concurrently, or one that updates the status displayed for the auction that the user is currently interested in, live, even in the middle of performing other operations - like bidding on that same auction.

In this exercise, you will write a client for the `AuctionManager` web service that issues a request to place a bid on an auction, without blocking while the request to place that bid is processed on the server side. This is the kind of more sophisticated logic that could be required for the two scenarios presented above.

1. Deploy the `EJBWebServices` project once again (you should have undeployed it at the end of the previous exercise).

2. Create a new Java Application project, call it `AsyncClients`, to avoid confusion with the other, simpler clients. This project, too, is standalone – any artifacts that this web service client will need will be generated by JAX-WS, given the WSDL description for the service.

3. Introduce a web service client reference to the `AuctionManager` web service. When you do this, NetBeans will configure this project to generate all the necessary client-side Java artifacts for this web service.

4. This client will need to specify that JAX-WS should generate artifacts that support the new asynchronous API supported by JAX-WS. The simplest way to do this is to use the NetBeans wizard edit to the web service client's attributes, to add this support.

5. Create a new application client for the `AuctionManager` web service, call it `AsyncPlaceBid.java`.

6. Write the client so that it places a bid on an Auction by calling the `placeBid` operation on the `AuctionManager` web service – but have the client perform some work, in between the request that the service place a new bid,

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Aula Matriz S.A.S.

and the display of the result of that request. It is enough that the server print out a message, between requesting the placing of the bid and displaying the response.

7. Run the client application, to test this asynchronous interaction.

There are a number of ways to capture the requirement the the client be able to use the asynchronous call API:

- Modify the server's WSDL description to include the requirement that all clients support the JAX-WS asynchronous API, if possible.
- Copy the server's WSDL description for this new client to refer to, locally. Modify this client's copy to include the requirement that JAX-WS support asynchronous interactions.
- Write a WSDL customization XML file so that the client's runtime add the requirement that JAX-WS support asynchronous interactions to the WSDL description received from the web service provider.
- Use the IDE to add the requirement that the client support asynchronous interactions to this web service client's artifacts.

Task 2 – Shut Down Web Services

Undeploy the web service provider application(s) you deployed in this exercise, and shut down the application server.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.
Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Aula Matriz S.A.S.

Exercise Summary

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

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Aula Matriz S.A.S.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Aula Matriz S.A.S.

Lab 7

Introduction to RESTful Web Services

Objectives

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

- Understand what RESTful Web Services are.
- Understand the five principles behind RESTful Web Services.
- Understand the advantages and disadvantages of a RESTful approach.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Aula Matriz S.A.S.

Exercise 1 – Understanding RESTful web services

1. Indicate whether each of the following assertions is true or false:
   (a) RESTful web services are procedural, rpc-oriented services.
   (b) RESTful web services prefer to minimize the number of interactions between client and server, and so prefer to deliver complex data representations.
   (c) RESTful web services commit to XML-based representations for communications.
   (d) RESTful web services prefer stateless interactions.
   (e) RESTful web service requests represent the operation to be performed somewhere in the URI used to invoke that operation on the server side.
2. Choose the statement that best describes the RESTful approach to choosing the right HTTP method to use for any request:
   (a) `GET` is always the preferred method to use.
   (b) `POST` is to be preferred over `PUT`, as it is more flexible.
   (c) `PUT` is to be preferred over `POST`, as it is more precise.
   (d) `POST` and `PUT` have different semantics, so the choice must be based the semantics of the operation.
3. Fill in the HTTP method that corresponds to each of the semantics quoted:

Method	Purpose
______	Read, possibly cached
______	Update or create without a known ID
______	Update or create with a known ID
______	Remove

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Aula Matriz S.A.S.

Lab Summary

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

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Aula Matriz S.A.S.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Aula Matriz S.A.S.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Lab 8

RESTful Web Services: JAX-RS

Objectives

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

- Implement RESTful web services using JAX-RS
- Deploy REST web services using Jersey, an implementation of JAX-RS

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Aula Matriz S.A.S.

Exercise 1 – Understanding How to Define Web Services Using JAX-RS

1. Indicate whether each of the following statements is true or false:

   (a) Every JAX-RS root resource class must be annotated with a `@Path` annotation.

   (b) If a public method of a resource class does not include an explicit HTTP method annotation, JAX-RS will act as though the code had specified `@GET`.

   (c) Just like web containers do with servlets, the JAX-RS runtime will create an instance of a root resource class the first time the runtime needs that class to handle a client request, then reuse that same instance as other clients require access to the same root resource class.

   (d) A class that inherits from `Application` is mandatory for every server-side application based on JAX-RS, so as to configure the set of resources offered by that application.

   (e) JAX-RS guarantees thread-safety for root resource classes annotated with the `@Singleton` annotation.

2. Two resource methods on the same resource class must differ in one of the following: (Choose all that apply)

   (a) `@Path` URI template.

   (b) HTTP Method.

   (c) Representations produced.

   (d) Representations consumed.

   (e) Visibility.

   (f) Arguments (in number or types).

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Aula Matriz S.A.S.

3. Choose the name of the the type of object that you can inject into a JAX-RS resource class to help it build URIs relative to the current one, to be used as part of responses. (Choose one).

   (a) `PathInfo`

   (b) `UriInfo`

   (c) `Context`

   (d) `Response`

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Aula Matriz S.A.S.

Exercise 2 – Create, Deploy, and Test the ItemManager Web Service

In this exercise, you're going to create, deploy, and test a simple RESTful web service for the Auction application, as a counterpart to the first simple JAX-WS service you implemented before. Once you have your web service ready, you will deploy it to a stand-alone Java VM. Since this is a RESTful web service, you can test it simply using a browser.

Task 1 – Define URIs Required by RESTful Service

The simplest service one could imagine using in the Auction project might be an `Item` manager web service, which would allow its clients to create/find/-manage the set of persistent `Item` descriptions used by the system, along the same lines of the JAX-WS `ItemManager` service you wrote before.

1. In a RESTful architecture, the first thing to do is identify the URIs that will be used to refer to the entities to be exposed through the RESTful service. Remember that URIs are to be used to identify the entity of interest, not the operations to be performed with that entity.

2. You do have to choose what operations will be supported by this RESTful service – but in a RESTful architecture, those operations have to be mapped to the standard HTTP methods that one could invoke on of the entities exposed through this service.

Table 8.1 will allow you to capture the RESTful request that will represent the operations that your web service will support.

Consider whether the requests that will accept parameters should describe them as components in the URI path, or query or form parameters to the request.

Task 2 – Define POJO Class

You will implement a JAX-RS `ItemManagerRS` service. It will be responsible for creating new `Item` instances, and finding existing `Item` instances that match a new item. It could also remove instances that are no longer used – but that is a more dangerous service to offer, within the Auction system.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Oscar Fernando Olivia Aragon (fernolivia@hotmail.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Aula Matriz S.A.S.

Request	HTTP Method	URI
List all Items		
Add New Item		
Update Item		
Get Item		
Remove Item		

Table 8.1: Requests Supported by RESTful Item Manager

Since you have not yet seen how to pass complex argument into and out of RESTful web services, limit your API to accept primitive types (and Strings), and to return the same: for example, the function to add a new `Item` could return the new `Item`'s id, and the function to remove an `Item` could take the id of the `Item` to remove. You may want to wait to implement the operation to find an `Item` until a later lab.

1. Create a new Java Application project in NetBeans. Call it `RESTfulServices`. Switch to it. You will use this project to create standalone JAX-RS (RESTful) web services.

2. Add the `JAX-RS 1.1` and `Jersey 1.1 (JAX-RS RI)` libraries to the project, to have access to the JAX-RS APIs and Jersey implementation.

   You will also need to add the dependencies that are required to support server-side operations for the Auction application.

3. Create a new class `ItemManagerRS`, in package `labs`. Define the functions that will be required to support the operations you identified above. Implement the business logic associated with each of the functions in the interface for `ItemManagerRS`.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Aula Matriz S.A.S.

NetBeans includes a wizard to create JAX-RS (RESTful) web services from scratch – but it is only available in enterprise applications. Since your first project will deploy simple standalone JAX-RS services, you will not be able to call on this NetBeans wizard until a later lab.

Task 3 – Incorporate JAX-RS Annotations

Once the `ItemManagerRS` is implement and working properly, it is time to turn it into a RESTful web service provider. Incorporate JAX-RS annotations, to turn your `ItemManagerRS` into a RESTful web service:

1. A `@Path` annotation at class level, to indicate that `ItemManagerRS` is a JAX-RS root resource .
2. Add a `@Path` annotation at method level, on each of the methods that is responsible for one of the operations offered by this RESTful web service, to indicate the URI template that triggers calls to this method.
3. Add an HTTP method annotation on each of the methods responsible for one of the operations offered by this service, to indicate which HTTP method is associated with each of these operations.
4. Add JAX-RS parameter annotations, if needed, to pass parameters encoded as part of the request into the web service method call.

Task 4 – Deploy Web Service to Standalone JVM

Deploy your `ItemManager` web service in a standalone JVM:

1. Introduce a new class to be your service runner – call it `Runner` – and implement it so as to deploy your web service.
2. Ensure that the database server is running.
3. Execute the `Runner` application just created.

Task 5 – Test Web Service Using Browser

RESTful web services represent web service interactions in terms of simple HTTP requests and responses. For simple kinds of interactions (those that are

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Aula Matriz S.A.S.

HTTP `GET` requests, or simple `POST` requests), a web browser is enough to generate the requests that the RESTful web service will process.

Use a browser window to generate `GET` requests to test your RESTful web service, according to the URI templates and HTTP methods you enumerated in Exercise 2.

Task 6 – Shut Down Web Service

Shut down the `Runner` application.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Aula Matriz S.A.S.

Exercise 3 – Create, Deploy, and Test the UserManager Web Service

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

In an earlier lab, you built a `UserManager` web service using the “contract-first” approach. A RESTful architecture does not have an equivalent notion – there is no standard description of what a web service will provide, written independently of and before implementing the web service.

In this exercise, you will implement the RESTful counterpart to the `UserManager` service you implemented before. The activities that the web service would need to support, as far as user management is concerned, include:

- Add new users to the application. This might involve specifying the user’s login id, full name, and email address. It could also include a password, to authenticate the user.
- Update information about an existing user, such as their full name, or their email address.
- List all users, or just the users that match some criteria.
- Remove users from the application.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

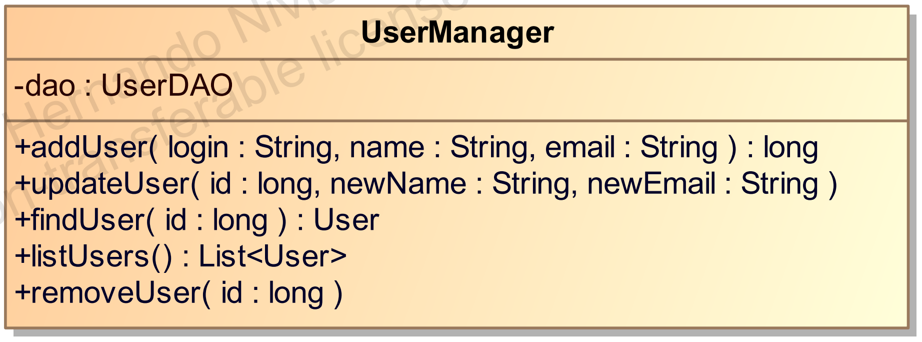

UserManager
-dao : UserDAO
+addUser(login : String, name : String, email : String) : long
+updateUser(id : long, newName : String, newEmail : String)
+findUser(id : long) : User
+listUsers() : List<User>
+removeUser(id : long)

Figure 8.1: Possible UML Diagram for `UserManager` Class

Figure 8.2 illustrates what the UML for a class that provides such operations could look like.

Since you have not yet learned how to pass complex types to or from a RESTful web service, you can delay support for listing all users until a later lab.

Aula Matriz S.A.S.

Request	HTTP Method	URI
List all Users		
Add New User		
Update User		
Get User		
Remove User		

Table 8.2: Requests Supported by RESTful User Manager

Task 1 – Define URIs Required by RESTful Service

The `UserManagerRS` web service will allow its clients to create/find/manage the set of persistent `User` instances representing users known by the system:

1. In a RESTful architecture, the first thing to do is identify the URIs that will be used to refer to the entities to be exposed through the RESTful service. Remember that URIs are to be used to identify the entity of interest, not the operations to be performed with that entity.

2. You do have to choose what operations will be supported by this RESTful service – but in a RESTful architecture, those operations have to be mapped to the standard HTTP methods that one could invoke on of the entities exposed through this service.

Table 8.2 will allow you to capture the RESTful request that will represent the operations that your web service will support.

Consider whether the requests that will accept parameters should describe them as components in the URI path, or query or form parameters to the request.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Aula Matriz S.A.S.

Task 2 – Define POJO Class

1. Create a new class `labs.UserManagerRS`, in the `RESTfulServices` project. Define the functions that will be required to support the operations you identified above.

2. Implement the business logic associated with each of the functions in the interface for `UserManagerRS`.

Task 3 – Incorporate JAX-RS Annotations

Once the `UserManagerRS` is implement and working properly, it is time to turn it into a RESTful web service provider. Incorporate JAX-RS annotations, to turn your `UserManagerRS` into a RESTful web service:

1. Add a `@Path` annotation at class level, to indicate that `UserManagerRS` is a JAX-RS root resource .

2. Add a `@Path` annotation at method level, on each of the methods that is responsible for one of the operations offered by this RESTful web service, to indicate the URI template that triggers calls to this method.

3. Add an HTTP method annotation on each of the methods responsible for an operation offered by this service, to indicate which HTTP method is associated with each of these operations.

4. Add JAX-RS parameter annotations, if needed, to pass parameters encoded as part of the request into the web service method call.

Task 4 – Deploy Web Service to Standalone JVM

Deploy your `UserManagerRS` web service to a standalone JVM. There are at least two ways to do this:

- Add a `main()` function to your class, and use it to deploy your web service.

- Introduce a new class to be service deployer – call it `Runner` – and implement it so as to deploy your web service.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Aula Matriz S.A.S.

Note that, if you implemented a class `Runner` separate from your `ItemManagerRS` in order to deploy that service in the earlier exercise, there may be nothing else to do now: the `Runner` class might have just relied on the JAX-RS runtime to find and deploy the root resources available to the application; if you simply run that application again now, it will find both root resources in this project, and it will deploy both.

Task 5 – Test Web Service Using Browser

RESTful web services represent web service interactions in terms of simple HTTP requests and responses. For simple kinds of interactions (those that are HTTP `GET` requests, or simple `POST` requests), a web browser is enough to generate the requests that the RESTful web service will process.

Use a browser window to generate `GET` or `POST` requests to test your RESTful web service, according to the URI templates and HTTP methods you enumerated in Exercise 2.

Task 6 – Shut Down Web Service

Shut down the `Runner` application.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Aula Matriz S.A.S.

Lab Summary

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

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Aula Matriz S.A.S.

Lab 9

JAX-RS Web Service Clients

Objectives

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

- Understand how to create web service clients using JAX-RS.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Oscar Hernando Nivia Aragon (olivia@hotmail.com) has a non-transferable license to use this Student Guide.

Aula Matriz S.A.S.

Exercise 1 – Understanding RESTful clients

1. Indicate whether each of the following statements is true or false:

   (a) Clients of a RESTful service that are implemented using the class `java.net.URL` in Java are limited to HTTP `GET` requests, because the `URL` class has that limitation.

   (b) `java.net.URL` cannot pass an entity body as part of any request.

   (c) The machinery associated with `java.net.URL` can automatically encode and decode query parameters and returns encoded in `Base-64`.

   (d) The machinery provided by the Jersey Client API can automatically encode and decode query parameters and returns encoded in `Base-64`.

2. Choose the type used to represent RESTful resources in the Jersey Client API: (Choose one).

   (a) `Resource`

   (b) `Client`

   (c) `WebResource`

   (d) `URL`

3. To obtain metadata about the most recent interaction with the web service, the Jersey Client API requires that:

   (a) The client specify that the expected type for the incoming payload be `ClientResponse`.

   (b) The client call a `getMetadata()` API in `WebResource`.

   (c) The client call the `WebResource` HTTP method within a `try/catch`, because the metadata will be part of the exception that the Jersey runtime will throw to the application.

   (d) The client gain access to the underlying `HTTPUrlConnection` instance because HTTP metadata is not exposed through the Jersey Client API.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Aula Matriz S.A.S.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Exercise 2 – Building Web Service Clients

You have implemented two different RESTful web services: `ItemManagerRS` and `UserManagerRS`. In this lab, you will build web service clients for those web services as stand-alone applications, using the Jersey client APIs.

Task 1 – Create New Project

To avoid confusion, create a new Java Application project (call it `RSClients`), to place your new standalone client applications into. This new project should depend on the `AuctionApp` project, since JAX-RS does not generate any artifacts for you: if a web service were to require, or return, an application type, you would need to provide that type. The project will also need to depend in the `JAX-RS 1.1` and `Jersey 1.1 (JAX-RS RI)` libraries.

At the moment, this may not be important, since your web services may be written in terms of parameters and return types that are primitives or instances of `String`. In general, however, you would be communicating with these web services using application types.

Task 2 – Build a Web Service Client for ItemManagerRS

The `ItemManagerRS` web service offers several operations, like: add new items to the persistent store, and remove existing items. A single web service client that is a standalone application may require then several parameters: the first one could be the name of the operation to perform (one of `add`, or `remove`), the others any information that may be required by each of those operations.

1. Write a standalone Java application (call it `clients.ItemManagerClient`) that accepts parameters, and uses them to invoke the appropriate operation on the `ItemManagerRS` web service.

   To call the RESTful web service, use the Jersey client API. This involves:

   - Obtaining an instance of `Client`.
   - Obtaining instances of `WebResource` that represent the resources to contact.

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Aula Matriz S.A.S.

- Invoke operations on those resources through their `WebResource`.

As an alternative, you could write several standalone web service client applications, one for each operation supported by the `ItemManagerRS` web service: each client application would then be hardcoded to invoke a specific operation.

2. Run the client application(s), to test all the operations in `ItemManagerRS`.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Aula Matriz S.A.S.

Task 3 – Build a Web Service Client for UserManager

The `UserManagerRS` web service offers several operations, like: add new users to the persistent store, and remove existing users.

1. Write a standalone Java application (call it `UserManagerClient`) that accepts parameters, and uses them to invoke the appropriate operation on the `UserManagerRS` web service – along the lines of the client application(s) you wrote earlier, to interact with the `ItemManagerRS` web service.

2. Run the client application(s), to test all the operations in `UserManagerRS`.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Aula Matriz S.A.S.

Exercise Summary

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

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Aula Matriz S.A.S.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Lab 10

JAX-RS and JavaEE

Objectives

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

- Deploy RESTful web services to a web container.
- Secure RESTful web services in a web container, delegating authentication to the container.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Aula Matriz S.A.S.

Exercise 1 – Understanding RESTful resources in a container

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

1. Indicate whether each of the following statements is true or false:

   (a) Deploying RESTful JAX-RS resources in a web application requires changes to the `web.xml` configuration file.

   (b) Deploying RESTful JAX-RS resources in a web application requires the presence of a subclass of `Application`.

   (c) Security constraints that apply to a JAX-RS resource can only be captured in the `web.xml` configuration for the web application that the resource is deployed in.

   (d) Declarative annotation of security constraints as annotations on the resource class automatically configures the web container's login configuration for this application.

   (e) Declarative authorization constraints can be specified in terms of specific users of the application.

   (f) The JAX-RS and JAX-WS specifications are not compatible, so the only way to have web services that can be reached via either REST or SOAP is to have two separate implementation classes, one implemented in JAX-RS, and the other in JAX-WS.

   (g) A JAX-RS resource that is implemented as a Singleton EJB must be made thread-safe programmatically at application level.

2. Choose the types that can be injected into a JAX-RS resource class to obtain information about the caller of a service: (Choose as many as appropriate).

   (a) `SecurityContext`

   (b) `ServletContext`

   (c) `UriInfo`

   (d) `ServletRequest`

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Aula Matriz S.A.S.

Exercise 2 – Deploy RESTful Web Services to a Web Container

In Lab 8, you built two simple web services, `ItemManagerRS` and `UserManagerRS`. They were both designed to be simple RESTful POJO web services, and they were both deployed standalone in a JVM. In this exercise you will deploy them into a web application, so as to leverage the functionality provided by the web container: you will be able to authenticate clients of the web service by delegating the authentication logic to the web container.

Task 1 – Copy Existing Web Services to Web Application

RESTful POJO web services can be deployed either standalone, using the HTTP server built into Java SE 6, or to any web container. The implementation of the POJO web service itself does not have to change, due to the environment it will be deployed into – any differences at that level are handled by the JAX-RS (Jersey) runtime.

1. To avoid confusion, create a new Web Application project (give it the name `RSInContainer`) to deploy the two web services into. This project will need to depend on the `AuctionApp` project, and all the usual suspects, for a server-side application.
2. Add to this project two server resource configuration entries:
   (a) Configure a database connection pool, so that this application (and, in particular, the persistence layer that your services rely on) can access the database efficiently. You can name this pool `auctionDbPool`, and use the NetBeans connection to the Auction database that you configured in in the setup lab, so that NetBeans can configure the connection pool.

   (b) Configure a JDBC resource, so that components within this project can access the database through the `auctionDbPool` via JNDI lookup. You should name this JDBC resource `jdbc/auction`.

   You can refer to the instructions for Lab 4 Exercise 2 Task 1: you had to set up a similar configuration for another web application then.
3. Copy the code for the two services `ItemManagerRS` and `UserManagerRS` from the `RESTfulServices` project you created in Lab 8 into this new project.

   The simplest way to copy the two services is:

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Aula Matriz S.A.S.

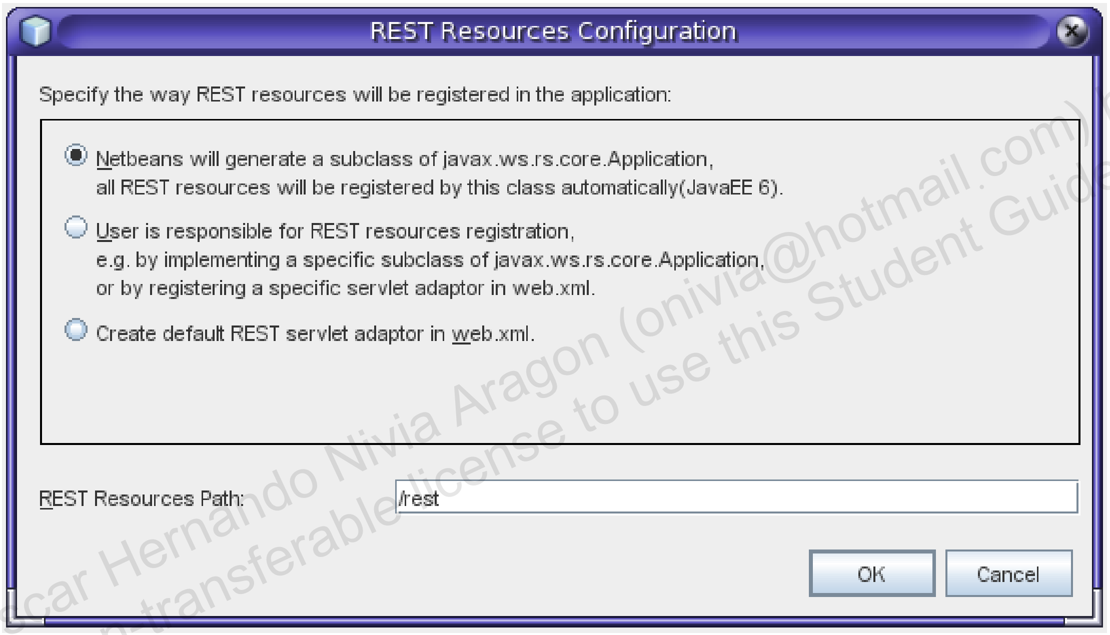
(a) Select the `labs` package from the original `RESTfulServices` project.

(b) Right-click on that `labs` node, and select `Copy`.

(c) Select the `Source Packages` node within the `RSInContainer` project.

(d) Right-click on that `Source Packages` node, and select `Paste`.

When you copy the two JAX-RS implementation classes into the `RSInContainer` project, NetBeans will note that you are importing JAX-RS resources: a dialog will pop up, asking you how NetBeans should proceed with the configuration of those resources within the web application. Figure 10.1 illustrates the dialog that you will see.


REST Resources Configuration

Specify the way REST resources will be registered in the application:

(●) Netbeans will generate a subclass of javax.ws.rs.core.Application, all REST resources will be registered by this class automatically(JavaEE 6).

( ) User is responsible for REST resources registration, e.g. by implementing a specific subclass of javax.ws.rs.core.Application, or by registering a specific servlet adaptor in web.xml.

( ) Create default REST servlet adaptor in web.xml.

REST Resources Path: /rest

Figure 10.1: Automatic JAX-RS Configuration Dialog on Import

You should go ahead and let NetBeans configure these resources automatically – just change the resource path that it will configure the resources with to be `/rest`, to match the instructions in the next step.

A new node named `RESTful Web Services` will appear in the `RSInContainer` project: it contains descriptions of the RESTful services that NetBeans knows to exist within the project.

Task 2 – Configuring Web Services Within Web Application

No explicit configuration changes to the web application (that is, changes to the `web.xml` file) are needed – JAX-RS will configure URLs to access the web

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.
Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Aula Matriz S.A.S.

services within the application by default, based on their `@Path` annotations. The URLs will have, by default, the form:

```
http://localhost:8080/RSInContainer/pathInAnnot
```

However, once RESTful web services are part of a web application, it may be necessary to distinguish incoming URL requests that are supposed to be handled by JAX-RS services from URLs that are to be processed by the web application front end. An `Application` class can be used to capture explicitly the set of RESTful resources to be offered by the application; it can also be used to specify a prefix to URLs that are to be processed by those resources, to distinguish RESTful requests from web front end requests, through a `@ApplicationPath` annotation.

If you selected to let the user register JAX-RS resources when you copied the two resource classes into the application, you will need to :

1. Create a class that inherits from `Application` – you could call your subclass `labs.AppResources`. Implement it so as to specify the set of RESTful resources provided by this web application: the two root resources `ItemManagerRS` and `UserManagerRS`.

2. Annotate your child class with an `@ApplicationPath` annotation, to set the prefix that distinguishes RESTful requests to `/rest`.

If you let NetBeans register your resources automatically, then the class it created will be found within a node called `Generated Sources (rest)` within the `RSInContainer` project. It will also include an `@ApplicationPath` annotation, to specify the prefix that will distinguish URIs that refer to RESTful services within this application.

As a result of this `@ApplicationPath` annotation, URL requests for the two web services will have the form:

```
http://localhost:8080/RSInContainer/rest/pathInAnnot
```

Task 3 – Deploy and Test Services in Web Application

Deploying the `RSInContainer` web application to a web container will automatically deploy the RESTful web services contained within it.

To test the services deployed within this web application, you could do the same things you did to test the original standalone services in Labs 8 and 9:

- Use a web browser to issue `GET` and `POST` requests.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Aula Matriz S.A.S.

- Write simple Java client applications that use the Jersey Client API to invoke operations on these services.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Aula Matriz S.A.S.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Exercise 3 – Secure POJO Web Services in Web Container

Earlier, you deployed the two web services `ItemManagerRS` and `UserManagerRS` to a web container, as part of a web application. One of the advantages of doing this is that it becomes possible to secure those web services, delegating authentication to the web container. In this exercise, you will secure access to the `UserManagerRS`, so that only administrators can create new users.

Task 1 – Define Security Constraints on Web Service

You have to capture, as part of the description of the service, that there are two kinds of users of the `UserManagerRS` web service, "administrators" and everyone else, and that only administrators are allowed to add new users to the application.

The way to configure security constraints in JAX-RS web services borrows the same annotations that JAX-WS (and before it, EJBs) use to capture these constraints.

You could also log the name of the person adding a new user to the container's log file, just to show that authenticated user information is provided to the web service.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Task 2 – Define Security Constraints on Web Application

Capturing authentication constraints at the level of the web service implementation classes isn't enough. JAX-RS will delegate authentication to the web container – but the web container requires additional configuration information, in order to perform authentication.

The configuration of the web application's `web.xml` to support authentication for JAX-RS web services is also a match for the configuration required for JAX-WS web services – with one exception: JAX-RS web services can be triggered by requests using any of the available HTTP methods.

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Aula Matriz S.A.S.

Task 3 – Configure Users in Authentication Realm

The actual authentication of principals is outside the scope of the specification of a web application – it is a responsibility of the web container. Therefore, configuration of the information associated with known principals (for example, their user names and password) must be done at the level of the web container.

Since you already configured a number of users in an earlier lab, let's just continue to use the same set of users as before.

Task 4 – Building Authenticating Web Service Clients

In an earlier lab, you created two standalone applications that use the Jersey client API to interact with RESTful services, which you may have called `ItemManagerClient` and `UserManagerClient`. You could use those two applications to test the web services that can be deployed as part of the `RSInContainer` web application – if they would just authenticate themselves to the web service:

1. Copy the two client applications into two new applications, which you could call `AuthItemManagerClient` and `AuthUserManagerClient`.

2. Modify both client applications so that the URI used to contact the web service matches the URIs associated with the web services deployed to the `RSInContainer` web application.

3. Modify the two new applications so that they provide authentication information, when they issue requests to their web services.

Task 5 – Deploy and Test Services in Web Application

1. Deploy the `RSInContainer` web application. This will deploy the web application, including all RESTful web services configured by the `Application` subclass in that web application.

2. Run your two authenticating client applications, to test that the web services are working correctly, and limiting access to the right users.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Aula Matriz S.A.S.

Lab Summary

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

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Aula Matriz S.A.S.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Aula Matriz S.A.S.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Lab 11

Implementing More Complex Services Using JAX-RS

Objectives

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

- Pass complex objects to and from a web service.
- Map Java exceptions from a web service provider to SOAP faults.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Aula Matriz S.A.S.

Exercise 1 – Understanding Features of JAX-RS

1. Indicate whether each of the following statements is true or false:

   (a) Jersey only supports XML representations out of the box, through Java's built-in JAXB technology. Other representations have to be added as application-level extensions, via classes that implement the two JAX-RS interfaces `MessageBodyReader` and `MessageBodyWriter`.

   (b) Any one resource method can only support one representation type.

   (c) `Response` objects are used to capture metadata about the return value to be delivered to the caller.

   (d) A resource method that needs to return metadata to a caller, when an exception is thrown within the resource method, must be defined to return a `Response` instance – so the method can return the appropriate metadata.

   (e) When creating a custom representation for an application-level type, both `MessageBodyReader` and `MessageBodyWriter` implementations for that application-level type must be provided.

   (f) Instances of entity provider types – classes that implement the two interfaces `MessageBodyReader` and `MessageBodyWriter` – are created and thrown away after each use.

2. Choose the class-level annotation that must be used for any classes that implement `MessageBodyReader`, `MessageBodyReader`, or `ExceptionMapper`: (Choose one).

   (a) `@Context`

   (b) `@Path`

   (c) `@Provider`

   (d) `@Resource`

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Aula Matriz S.A.S.

3. How can you identify sub-resource locator methods? (Choose the one best match).

   (a) They are methods in a resource class that have no `@Path` annotation.

   (b) They are methods in a resource class that have no HTTP method annotation.

   (c) They are methods in a resource class that are annotated with `@Resource`.

   (d) They are private methods in a resource class.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Aula Matriz S.A.S.

Exercise 2 – Pass Complex Types

The two RESTful web service you have implemented so far, `ItemManagerRS` and `UserManagerRS`, have been limited by the fact that you had only seen how to pass simple types. In this Module, you learned how to pass complex types, and so you can now make these two services more complete.

You may work on either the `RESTfulServices` project (for standalone POJO web services deployed to a JVM), or the `RSInContainer` project (for POJO web services deployed to a web container).

Task 1 – Complete the API of the Existing Web Services

Complete the APIs of your two web services, to include those operations that require complex types as output parameters:

1. Add support for a `findItem` operation to your `ItemManagerRS`, which accepts an id as a parameter, and returns the `Item` with that id.
2. Add support for a `findUser` operation to your `UserManager`, which accepts an id as a parameter, and returns the `User` with that id.

When you first defined the two services, you considered what URL templates and HTTP methods would be appropriate for each of the operations that the RESTful service would support on the resources that would be offered. The methods that you add here will need to be annotated with `@Path` and parameter annotations to match.

Task 2 – Update Client Applications

You will need to update your client applications to include support to invoke the new operations just added to the `ItemManagerRS` and `UserManagerRS` web services.

Task 3 – Deploy and Test Services in Web Application

Deploy the updated web services, and test them using your updated client applications.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable licence to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Aula Matriz S.A.S.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Exercise 3 – Improve Web Service Responses

RESTful web services are expected to follow closely the model initially defined by the HTTP standards for web interaction. In particular, resource responses are expected to carry status codes along with any payload that may be intended for the client, in order to give the client more information (*meta-data*) about the request just processed.

In this exercise, you will update your JAX-RS `UserManagerRS` web service to provide that additional metadata:

- Requests to add users ought to return a `201` status code (“Created”), along with the URI where the client will find the entity just created, in the future. In addition, the server could return the actual entity created back to the client, as the payload of the return message – instead of just the id associated with the new entity, as you had done before.
- Requests to update users ought to return a `404` status code (“Not Found”) if the user that is supposed to be updated cannot be found.

  Such requests could also return a `304` status code (“Not Modified”) if the update operation did not actually change the user specified.

You may modify your `UserManagerRS` web service in either the `RSInContainer` or `RESTfulServices` projects. The version of `UserManagerRS` deployed as part of the `RSInContainer` web application already has some of this more sophisticated metadata – the authentication support provided through the web container will result in responses with a `401` (“Not Authorized”) status code – so it may be a better choice for enhancing even further.

1. Modify the implementation of the method in `UserManagerRS` that handles requests to add new users so that it returns the `201` status code, along with the location of the new resource.
2. Modify the implementation of the method in `UserManagerRS` that handles requests to update users so that it returns “Ok” or “No Found”, as appropriate. You may also have it return “Not Modified”, when applicable.

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Aula Matriz S.A.S.

Lab Summary

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

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Aula Matriz S.A.S.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Lab 12

Trade-Offs Between JAX-WS and JAX-RS Web Services

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

- Discuss the trade-offs involved in the choice to implement a web service using either JAX-WS or JAX-RS technology.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Aula Matriz S.A.S.

Exercise 1 – Understanding the difference between JAX-WS and JAX-RS technologies

1. Choose the qualifier that best describes each of the two types of web services in the table below. (Choose one row for each column).

	SOAP services	**REST services**
Activity-oriented		
Method-oriented		
Resource-oriented		
URI-oriented		

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Aula Matriz S.A.S.

Exercise 2 – Building More Complex Web Services Using JAX-RS

The web services you are implementing are there to support an Auction application. In this exercise, you will write some of the higher-level services associated with such an application, as RESTful services.

For reference purposes, Figure 12.1 presents a possible UML diagram for the JAX-WS implementation of this service that you built earlier in the course. The RESTful implementation of the web service may organize its API differently – but the functionality ought to be equivalent to that of the earlier JAX-WS version of the service.

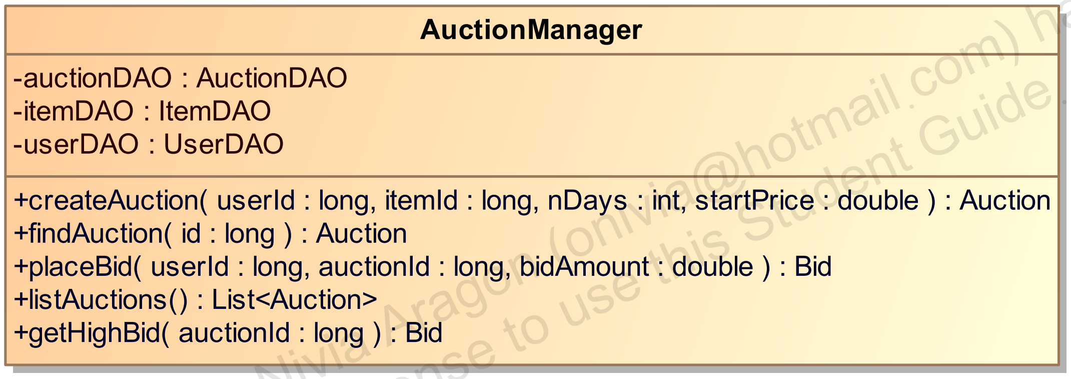

AuctionManager
-auctionDAO : AuctionDAO
-itemDAO : ItemDAO
-userDAO : UserDAO
+createAuction(userId : long, itemId : long, nDays : int, startPrice : double) : Auction
+findAuction(id : long) : Auction
+placeBid(userId : long, auctionId : long, bidAmount : double) : Bid
+listAuctions() : List<Auction>
+getHighBid(auctionId : long) : Bid

Figure 12.1: Possible UML Diagram for AuctionManager Web Service

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Task 1 – Define URIs Required by RESTful Service

The goal for the Auction application was to support the ability to manage auctions – so you need to provide an Auction service, which you will now implement as a RESTful service. An Auction application must support, at a minimum, the ability for users to create and look up auctions, and place bids on existing auctions.

1. In a RESTful architecture, the first thing to do is identify the URIs that will be used to refer to the entities to be exposed through the RESTful service. Remember that URIs are to be used to identify the entity of interest, not the operations to be performed with that entity.

2. You do have to choose what operations will be supported by this RESTful service – but in a RESTful architecture, those operations have to be

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Aula Matriz S.A.S.

mapped to the standard HTTP methods that one could invoke on of the entities exposed through this service.

Consider whether the requests that will accept parameter should describe them as components in the URI path, or query or form parameters to the request.

Task 2 – Implement a AuctionManager Web Service

1. Define a `AuctionManager` Web Service; you could call it `AuctionManagerRS`.
2. Include support for the operations you identified above.

   The `AuctionManagerRS` web service must be implemented as a JAX-RS root resource, with the proper `@Path` and HTTP method annotations on each of the methods that will support those RESTful operations.

Task 3 – Implement a RESTful Client Application

You may be able to test some of the methods on this new Auction service simply by using a browser. In order to test all the services, however, it may be best to implement a client application that uses the Jersey client API to invoke all the operations provided by the `AuctionManagerRS` web service.

Task 4 – Deploy and Test Services in Web Application

Deploy the new Auction web service, and test it using your auction client application.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Aula Matriz S.A.S.

Lab 13

Web Services Design Patterns

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

- Decide when to apply web services-based design patterns .

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Aula Matriz S.A.S.

Exercise 1 – Understanding Web Services Design Patterns

1. Which pattern has the following design goals:
   - Decouple the input and the output messages.
   - Deliver the output message from the server to the client.
   - Associate an output message to the corresponding input message.

   Choose one:

   (a) Asynchronous Interaction

   (b) JMS Bridge

   (c) Web Service Cache

   (d) Web Service Broker

2. Consider a simple design of an application that wants to integrate two remote services into its own workflow. Such a simple design assumes that a transaction begun by the client automatically propagates to both remote services. This allows both remote services to join that same transaction. When they have done so, a failure during processing in the second remote service allows it to roll back its own state by rolling back the current transaction. However, this automatically rolls back the work done by the first remote service, and any work done by the client, during that same transaction.

   Which pattern would you implement in the above scenario to support the transactional behavior of the application? (Choose one).

   (a) JMS Bridge

   (b) Web Service Cache

   (c) Web Service Broker

   (d) Web Service Logger

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Aula Matriz S.A.S.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Lab 14

Best Practices and Design Patterns for JAX-WS

There is no exercise for this module.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Aula Matriz S.A.S.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Aula Matriz S.A.S.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Lab 15

Best Practices and Design Patterns for JAX-RS

There is no exercise for this module.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Aula Matriz S.A.S.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Aula Matriz S.A.S.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Level II

Detailed Labs

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Aula Matriz S.A.S.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Lab 1

Introduction to Web Services

Objectives

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

- Describe the characteristics of a web service.
- Describe the advantages of developing web services within a JavaEE container.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Aula Matriz S.A.S.

Exercise 1 – Describing Web Services

1. Which of the following are characteristics of a web service? (Choose all that apply)

   (a) Tightly-coupled

   (b) Loosely-coupled

   (c) Interoperable

   (d) Platform-specific

2. Indicate, for each of the following, whether they are a characteristic of traditional RPC solutions, or web services. (Write either "RPC" or "Web Services" on the line next to each)

   Local to an enterprise ________________

   Message-driven ________________

   Procedural ________________

   Firewall-friendly ________________

   Variable transports ________________

3. Mark in each cell in the table whether the web services technology listed exhibits the characteristic listed:

	SOAP	**REST**
Use XML		
Has Formal Definition		
Relies on HTTP		

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

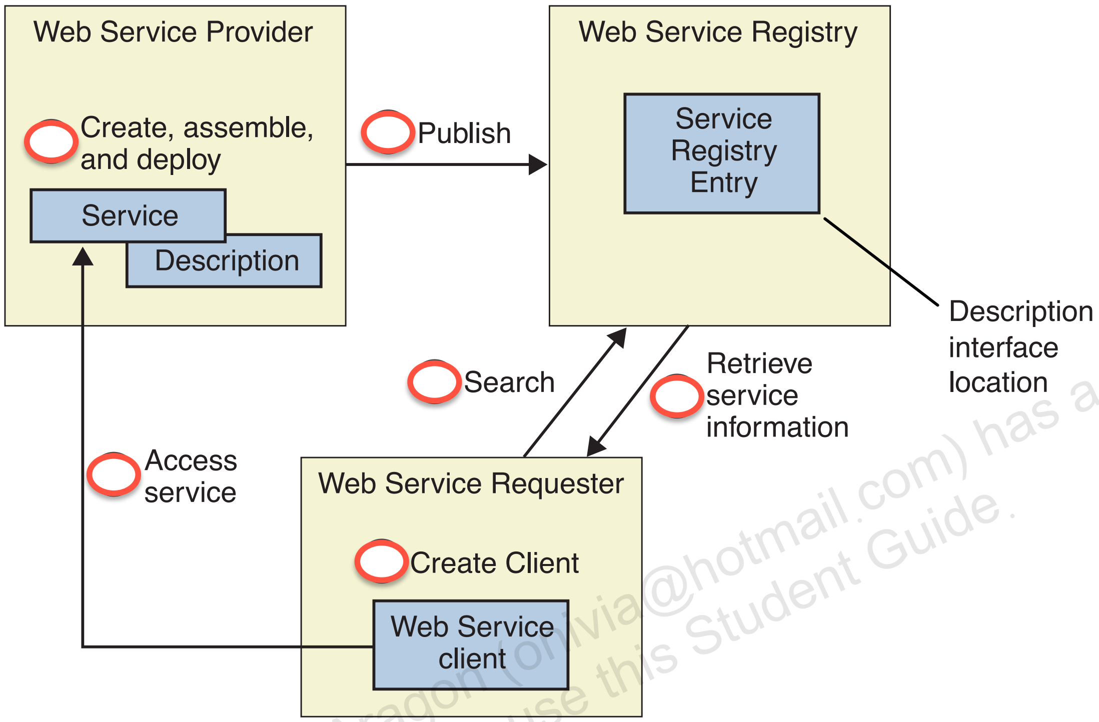
Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Aula Matriz S.A.S.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

4. Fill in the red ovals with numbers, indicating the order in which the steps illustrated in this figure are executed:



Oscar Hernando Olivia Aragon (olivia@hotmail.com) has a non-transferable license to use this Student Guide.

5. Select the features that correspond to support that the EJB component model offers web services developers. (Choose all that apply).

   (a) Programmatic transaction control.

   (b) Declarative transaction control.

   (c) Scalability through pooling.

   (d) Availability through robustness of Java implementation/runtime.

6. Which of the choices below constitutes deployment choices that are open to the web services deployer? (Choose all that apply).

   (a) Standalone Java applications on any JVM, because of web server machinery built-into JAX-WS and JAX-RS runtime.

   (b) Servlet endpoints deployed to a web container.

   (c) EJB service endpoints.

   (d) JMS-based runtimes, because the choice of a JMS transport is required by the web services specifications.

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Aula Matriz S.A.S.

7. Web services and their clients can interact with each other regardless of the platform on which they are running. This is possible in a SOA by means of (choose one):

   (a) Compatibility

   (b) Interoperability

   (c) Accessibility

   (d) Security

8. Which is the only characteristic of a service that a requesting application needs to know about in SOA? (Choose one).

   (a) Private interface

   (b) Protected interface

   (c) Public interface

   (d) Package interface

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Aula Matriz S.A.S.

Lab Summary

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

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Aula Matriz S.A.S.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Aula Matriz S.A.S.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Lab 2

Using JAX-WS

Objectives

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

- Create simple web services using JAX-WS:
  - Code-first, starting from Java classes
  - Contract-first, starting from WSDL descriptions
- Deploy web service providers using just JavaSE.
- Create and Run web service clients.

Oscar Hernando Nivia Aragon (ohnivia@hotmail.com) has a non-transferable license to use this Student Guide.

Aula Matriz S.A.S.

Exercise 1 – Understanding JAX-WS

1. Identify the correct tool to be used for generating JAX-WS artifacts in each of the following cases:

   (a) From a URL of a WSDL file

   (b) From a Java implementation source file

   (c) From a Java class file

2. Which is the annotation used to describe an argument of a web service operation. (Choose one).

   (a) `@WebResult`

   (b) `@WebService`

   (c) `@WebMethod`

   (d) `@WebParam`

3. Building web services according to the "contract-first" approach is considered a best practice because:

   (a) Writing WSDL descriptions first is more maintainable, because WSDL descriptions are more concise.

   (b) Writing Java code first is more flexible, because the designer can capture characteristics of the service in Java code that cannot be captured in WSDL.

   (c) Writing Java code first is safer, because the developer can capture constraints in Java code that cannot be captured in WSDL.

   (d) Writing WSDL descriptions first is safer, because the developer can capture constraints in WSDL descriptions that cannot be captured in Java code.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Aula Matriz S.A.S.

Exercise 2 – Create, Deploy, and Test a JAX-WS Web Service Code-First

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

In this exercise, you're going to create, deploy, and test a simple web service for the Auction application. You will implement this web service using a "code-first" approach, implementing the Java class that will be the service provider first, then turning it into a web service through the introduction of JAX-WS annotations. Once you have your web service ready, you will deploy it to a stand-alone Java VM, and then test it using the SoapUI plugin for NetBeans.

Task 1 – Define POJO Class

The simplest service one could imagine using in the Auction project might be an `ItemManager`, which would allow its clients to create/find/manage the set of persistent `Item` descriptions used by the system.

When sellers want to run auctions on the system, they have to enter a description for the item to be auctioned off - this is where `Item` instances come into play: instead of just creating a new Item description for each new auction started, the system wants to allow users to look for existing Items that match the new item the seller would want to put up for auction, so as to reuse that Item for as many auctions as are interested in that same description. Among other things, this might make it easier for the system to offer the ability to find similar auctions...

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

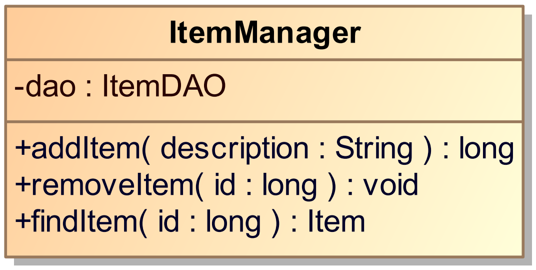

ItemManager
-dao : ItemDAO
+addItem(description : String) : long
+removeItem(id : long) : void
+findItem(id : long) : Item

Figure 2.1: Possible UML Diagram for `ItemManager` Class

The `ItemManager` service then would be responsible for creating new `Item` instances, and finding existing `Item` instances that match a new item. It could also remove instances that are no longer used – but that is a more dangerous service to offer, within the Auction system. For convenience, limit your API to accept primitive types (and Strings), and to return the same: for example, the function to add a new `Item` could return the new `Item`'s id, and the function

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Aula Matriz S.A.S.

to remove an `Item` could take the id of the `Item` to remove. You may want to wait to implement the operation to find an `Item` until a later lab – this function would return an `Item`, given its id.

Figure 2.1 shows what a UML diagram for `ItemManager` might look like.

1. Create a new Java Application project; call it `WebServices`. For convenience, you should place this new project in the `exercises` folder in your home directory – that way, it will be kept separate from the two folders, `examples` and `projects`, that contain all the code provided to you.

   You could skip the creation of a Main Class, immediately when the project is created.

```
Java Development
→ Java Application Projects
→ Creating Projects
```

2. Associate the existing `AuctionApp` project to your newly created project, so you can use all the types provided to you without modification. If you do not currently have the `AuctionApp` project open, go ahead and open it, first – it will be easiest to associate other open projects to your newly-created project.

   You will be writing new web services, so your `WebServices` project should depend on the two libraries for JAX-WS 2.2 and JAXB 2.2.

   Since the `WebServices` project will depend on `AuctionApp`, and it in turn uses JPA 2.0, your project will need to specify that it depends on JPA 2.0, as well – even though you may never refer to anything there explicitly. In NetBeans, the JPA 2.0 runtime is represented by the standard library named `EclipseLink (JPA 2.0)`.

   The same goes for the Derby (or Java DB) client library: since your project depends on `AuctionApp`, and it depends on `DerbyJDBC`, so will your project, too.

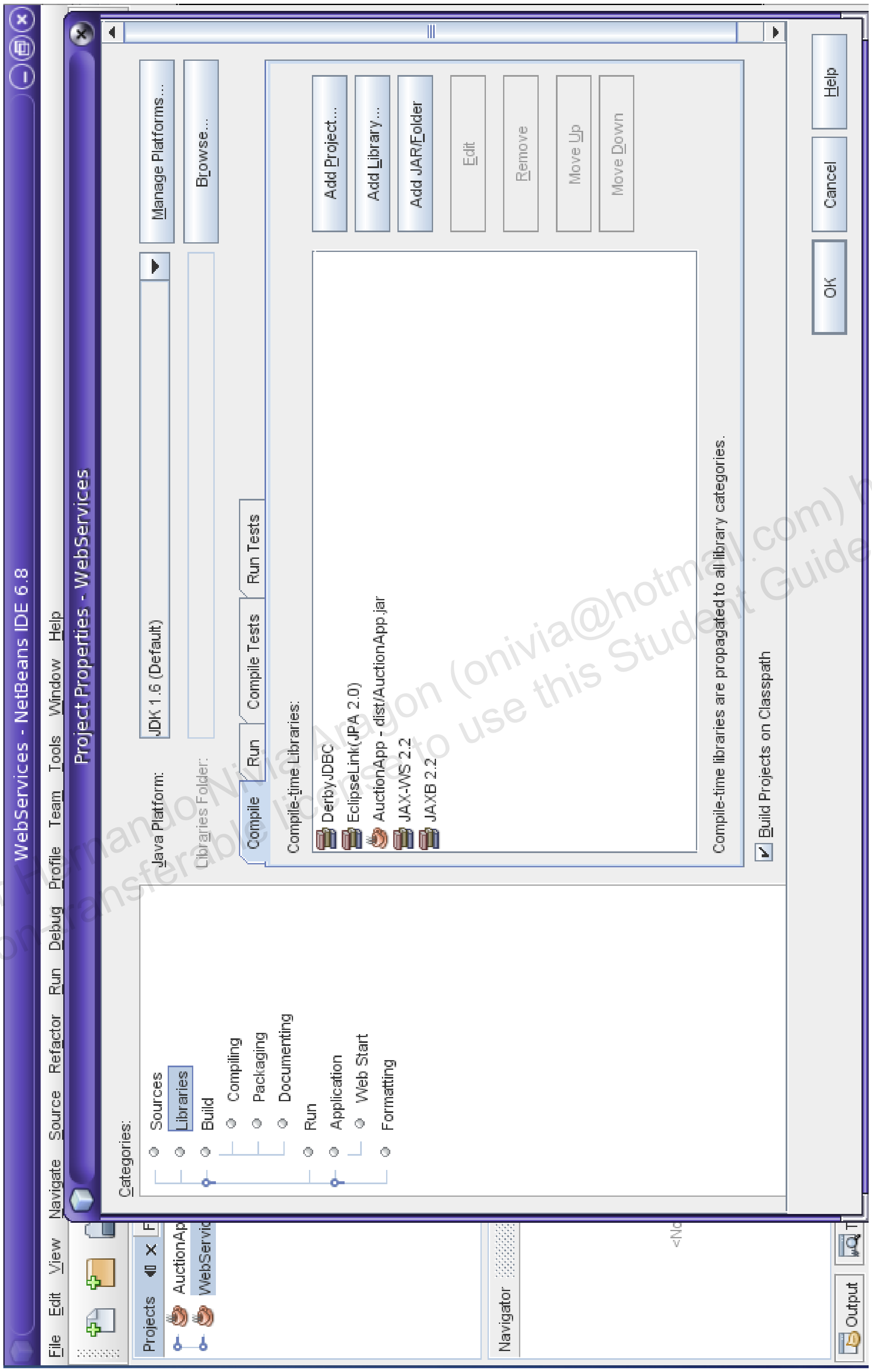
```
Java Development
→ Java Application Projects
→ Modifying Project Libraries
```

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Aula Matriz S.A.S.



Java Platform: JDK 1.6 (Default)

Libraries Folder:

Compile-time Libraries:

- DerbyJDBC
- EclipseLink(JPA 2.0)
- AuctionApp - dist/AuctionApp.jar
- JAX-WS 2.2
- JAXB 2.2

Compile-time libraries are propagated to all library categories.

[✓] Build Projects on Classpath

Figure 2.2: Project Properties for WebServices Project

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Oscar Fernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Aula Matriz S.A.S.

Figure 2.2 shows what the `Project Properties` dialog for your newly created project should look like, once you have added all the required dependencies.

3. Create a new class `ItemManager`, in package `labs`. Define as its public API the functions discussed above.

```
Java Development
→ Java Classes
→ Creating Java Classes
```

You could go ahead and create the package `labs` first, then add the new class `ItemManager` within it. However, the NetBeans new class wizard will allow you to enter a new fully-qualified class name; in that case, it will create the package, if necessary, in order to create the class within it.

As you use classes provided to you, you will probably want to `import` the corresponding fully-qualified class names. NetBeans can help you introduce the appropriate `import` using its `Fix Imports` wizard (available by right-clicking within the source code window), or via "quick fixes" (available by clicking on light-bulbs, when they appear on the left margin of the source window).

4. Implement the business logic associated with each of the functions in the interface for `ItemManager`.

The domain types you will need for the Auction domain, and in particular class `Item`, are provided to you in the `AuctionApp` project, along with a set of DAO types to manipulate persistent domain instances. By having your `WebServices` project depend on `AuctionApp`, all these types are available to you.

Your web service implementation classes should simply delegate to an instance of the appropriate DAO the work required to manage persistent instances of domain types.

Remember that the first parameter of the POJO DAOs provided by the `AuctionApp` project are meant for scenarios in which you will need to manage transaction boundaries yourself: for the time being, `null` value for this first parameter, to let the DAO manage transactions itself, would be appropriate.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Aula Matriz S.A.S.

NetBeans provides a wizard to create a new Web Service using the "code-first" approach – but only if the project it will belong to is an enterprise project (a web application, or enterprise application). Since you will be working with simple standalone Java projects, to start, you will not be able to take advantage of this wizard.

Task 2 – Incorporate JAX-WS Annotations

Once the `ItemManager` is implement and working properly, it is time to turn it into a web service provider. Incorporate JAX-WS annotations, to turn your `ItemManager` into a web service.

At a minimum, you will need to add a `@WebService` annotation to the `ItemManager` implementation class, to turn it into a web service provider.

Since JAX-WS is incorporated in Java SE 6, there should be no need to modify the classpath for the `WebServices` project to add support for it. However, Java SE 6 actually has one of a couple of different versions of JAX-WS built in, depending on which update is installed. To play it safe, make sure your project depends on the JAX-WS 2.2 (and JAXB 2.2) libraries provided by NetBeans.

Task 3 – Customize Project Build Script

Once we have modified `ItemManager` so that it can be used as as web service implementation class, we need to generate all the additional artifacts that JAX-WS requires in order to actually deploy the web service. This may require additional tasks in the project's `ant` script, `build.xml`.

The `build.xml` script for your project does not appear in the `Project` tab for your project in NetBeans, since the IDE considers that a private file that the developer should not have to edit. You can find it by switching to the `File` tab, where all the files that belong to a directory are presented.

The elements that need to be added to the NetBeans `build.xml` file are listed in Code 2.1. They can also be found in the `build.xml` file for the project named

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Aula Matriz S.A.S.

Managers in the examples/jaxws folder – it is the examples project that contains all the standalone JAX-WS examples used throughout the course.

```
<taskdef name="wsgen"
         classname="com.sun.tools.ws.ant.WsGen"/>
<target name="-post-init" depends="">
  <mkdir dir="generated"/>
  <mkdir dir="generated/xml"/>
  <mkdir dir="generated/src"/>
</target>
<target name="-post-compile" depends="-post-init">
  <wsgen
      sei="labs.ItemManager"
      destdir="build/classes"
      resourcedestdir="generated/xml"
      sourcedestdir="generated/src"
      keep="true"
      verbose="true"
      genwsdl="true"
      xendorsed="true"
      protocol="soap1.1">
    <classpath>
      <pathelement path="${build.classes.dir}"/>
      <pathelement
        location="${reference.AuctionApp.jar}"/>
    </classpath>
  </wsgen>
</target>
```

Code 2.1: Elements needed in build.xml for "Code-First"

Once you have modified your project's build.xml file, build the project once again. After the build completes successfully, you should find two new directories within your project:

- generated/src –
  This directory will contain the source code for the Java artifacts generated for your ItemManager web service.

- generated/xml –
  This directory will contain the WSDL/XML artifacts generated for your ItemManager web service.

In addition to these, the compiled versions of all Java artifacts will be placed in the build folder, along with all your other compiled classes.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Aula Matriz S.A.S.

Although the compiled code for the generated Java classes is present in the `build/classes` directory, the IDE may refuse to accept that these classed exist, when compiling your own code. To solve this issue, you should add the folder where the JAX-WS tools will place generated source classes (in this example, `generated/src`) as a *source folder* to the IDE. To add source folders to a project, right-click on your project, then choose `Sources` in the tab on the left. Initially, your project will have a single source folder called `src`. Add a second source folder for `generated/src`.

Task 4 – Deploy Web Service to Standalone JVM

Deploy your `ItemManager` web service to a standalone JVM. The simplest way to do this may be:

1. Create a new class, call it `ItemManagerRunner`, to be the application that deploys your web service. This new class will have a `main()` function.
2. In `main()`, create an instance of your web service implementation class `ItemManager`, and deploy it as a web service.

   Remember that this involves, at a minimum, two steps:

   (a) Create an instance of the web service implementation class.

   (b) Use the `Endpoint` class to `publish` the instance of the web service implementation class that you created at a particular URL.

   This URL needs to be the full URL that clients will use to contact the web service: it has to include protocol, hostname, port number, and the path that will identify the web service itself. You could use something as simple as:

   ```
   http://localhost:8081/ItemManagerService
   ```
3. Ensure that the JavaDB database engine is already running.
4. Run this application.

```
Java Development
→ Java Classes
→ Executing Java Programs
```

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Aula Matriz S.A.S.

As a simple test that your web service actually deployed successfully, remember that you can ask your web service to supply you with its WSDL description.

Task 5 – Test Web Service Using SoapUI

More through testing may be appropriate, however. Since we have not yet discussed how to create web service client applications that consume web services, we cannot write our own test client from scratch.

There is a tool we can use, instead, called `SoapUI`. It is available as a plugin for IDEs, or as standalone application – which you will find installed on your workstation. This tool will read the WSDL description of a web service, then generate a simple test harness that will help you write SOAP requests to deliver to the web service, then show you the return SOAP structure that your service delivers back to the calling client.

A quick tutorial on how to use `SoapUI` to test a web service can be found in the `resources` folder in your home directory, or directly from the authors' web site: `http://www.soapui.org/gettingstarted/your_first_soapUI_project.html`

The URL for the web service you would provide the `SoapUI` application with is constructed from the one that you indicated when publishing your service in the previous step – remember that you can add the suffix `?WSDL` to the web service URL, in order to request the WSDL description for that same service. You might use something like:

```
http://localhost:8081/ItemManagerService?WSDL
```

Task 6 – Shut Down Web Service

When you started your web service application running, NetBeans displayed an output panel that contains the output produced by that application. That same panel includes the controls required to shut down, or restart, that application. Figure 2.3 illustrates that output panel, including those two controls:

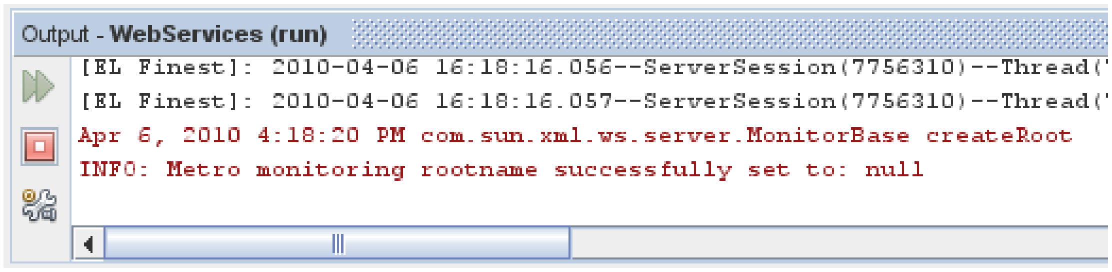
▶▶ Restart application   ■ Stop application

In addition to that, NetBeans includes GUI elements that advise the user that there are applications running within NetBeans, and which allow the user to

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.
Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Aula Matriz S.A.S.



```
[EL Finest]: 2010-04-06 16:18:16.056--ServerSession(7756310)--Thread('
[EL Finest]: 2010-04-06 16:18:16.057--ServerSession(7756310)--Thread('
Apr 6, 2010 4:18:20 PM com.sun.xml.ws.server.MonitorBase createRoot
INFO: Metro monitoring rootname successfully set to: null
```

Figure 2.3: Application Output for `WebServices` Project

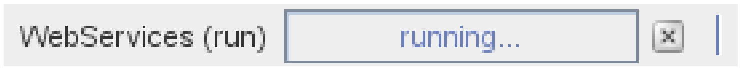


Figure 2.4: Process Indicators

shut them down. These elements appear at the bottom of the NetBeans GUI window, and are illustrated in Figure 2.4.

```
Java Development
→ Terminating a Running Process
```

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.
Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Aula Matriz S.A.S.

Exercise 3 – Create, Deploy, and Test a JAX-WS Web Service Contract-First

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

In this exercise, we're going to create, deploy, and test a simple web service for the Auction application. We will implement this web service using a "contract-first" approach, describing first the API that we intend for our web service to implement, using the WSDL vocabulary to create our description. Once the WSDL description is ready, we will then implement the Java class that will be the service provider. Once we have our web service ready, we will deploy it to a stand-alone Java VM, and test it using the SoapUI plugin for NetBeans.

Task 1 – Define Abstract Description for Web Service in WSDL

Another simple service that the Auction system will need is a `UserManager` service, which allows people to register with the system, maintaining persistent objects for each user that has registered. The activities that the application would need to support, as far as user management is concerned, include:

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

- Add new users to the application. This might involve specifying the user's login id, full name, and email address. It could also include a password, to authenticate the user.
- Update information about an existing user, such as their full name, or their email address.
- List all users, or just the users that match some criteria.
- Remove users from the application.

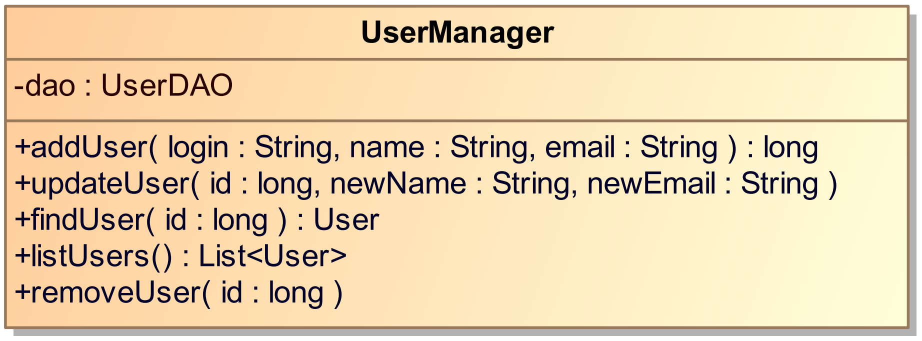

UserManager
-dao : UserDAO
+addUser(login : String, name : String, email : String) : long
+updateUser(id : long, newName : String, newEmail : String)
+findUser(id : long) : User
+listUsers() : List<User>
+removeUser(id : long)

Figure 2.5: Possible UML Diagram for `UserManager` Class

Figure 2.5 shows what a UML diagram for `UserManager` might look like.

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Aula Matriz S.A.S.

1. Identify the basic functions that such a `UserManager` would need to support, including their mode of interaction, and parameters and return types.

2. Write the abstract WSDL description for this web service. You could name the file `UserManager.wsdl`, in package `labs`. This abstract description is the one captured in the `portType` and `message` elements of the WSDL file, along with the proper XML schema types, if appropriate.

   Name the port type `UserManager`, to be consistent with the UML diagram shown above.

   Remember that a web service is described in a WSDL file in terms of `portType`, which will include a number of `operation` elements. Each of these represent an interaction between a client and the web service – but the type of interaction need to be described explicitly, in terms of `input` and `output` messages that capture the *message exchange pattern* associated with each operation.

   Input and output messages are described in WSDL files in terms of `message` elements. Each message element needs to describe the information that it will carry across; this description should be captured by XML schema types associated with each message element.

   You can split the description of the service into several files (XML schema, logical WSDL description, WSDL bindings) as shown during lecture. It is also the way the description for the `PassengerManager` service in project `Managers` is organized. Alternatively, you can just write a single description file.

   If you chose to separate logical from concrete description, at this point you should have three files:

   - `UserManager.xsd` –
     An XML Schema file that contains the definitions for all the information that will be carries by any messages between client and web service provider.

   - `UserManager.wsdl` –
     The logical description of the web service, in terms of the set of operations that this service will support. This description depends on the XML Schema types defined above.

   - `UserManagerSvc.wsdl` –
     The concrete description of the service, which includes the information necessary to actually communicate with the service. This description depends on the logical description defined above.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Aula Matriz S.A.S.

If you choose to capture all the elements that describe the `UserManager` service in a single XML file, name it `UserManagerSvc.wsdl` – just so that your exercise remain consistent with these instructions.

```
Java EE Development
→ Web Services
→ JAX-WS Web Services
→ Creating WSDL Files for JAX-WS Web Services
```

You can limit your API to functions that will only require or produce simple data. We will revisit this contract in a later chapter, once we learn how to write more complex WSDL files and their corresponding XML Schema types.

In principle, you should write the WSDL description for this new `UserManager` service from scratch. However, writing WSDL descriptions can be quite cumbersome, until one gains enough experience. You could start with a template of the description you will need to write, instead – grab a copy of the WSDL description for the `PassengerManager` service from the `Managers` example project, and use it as a starting point for your own.

If you do copy the WSDL description of the `PassengerManager` service, remember that you should rename the elements within that description: port type, operations, messages, schema types – but also the XML namespace used by the WSDL and XML Schema elements.
Also, the parameters to the `addUser` operation are going to be different from those that `addPassenger` required.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Task 2 – Incorporate Concrete Description for Web Service into WSDL

Complete the WSDL description for our new `UserManager` service by incorporating concrete details to the WSDL file. Remember that this concrete information is captured in the `binding` and `service` elements in the WSDL file.

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Aula Matriz S.A.S.

The kinds of information included as "concrete details" in a WSDL file include: the WSDL style (document literal vs. RPC literal), the address where web service will be found, and any `SOAPAction` header.

Task 3 – Customize Project Build Script

In the "contract-first" approach, there are more artifacts to be generated from the WSDL definition, and the actual generation has to be requested explicitly – in order to write any code in Java that refers to the service that was initially described contract-first, the developer will need a Java interface for that service, and the only way to get it is by running the proper code generator.

1. Modify your build script to include steps require to trigger the generation of artifacts before compilation of the source code for your project.

```
<taskdef name="wsimport"
         classname="com.sun.tools.ws.ant.WsImport"/>
<target name="-post-init" depends="">
  <mkdir dir="generated"/>
  <mkdir dir="generated/src"/>
</target>
<target name="-pre-compile" depends="-post-init">
  <wsimport
    wsdl="${basedir}/${src.dir}/labs/UserManagerSvc.wsdl"
    destdir="build/classes"
    sourcedestdir="generated/src"
    keep="true"
    verbose="true"
    extension="true"
    xendorsed="true"
    package="labs.generated">
  </wsimport>
</target>
```

Code 2.2: Elements needed in `build.xml` for "Contract-First"

The elements that need to be added to the NetBeans `build.xml` file are listed in Code 2.2. They can also be found in the `build.xml` file for the project named `Managers` in the `examples/jaxws` folder – it is the examples project that contains all the standalone JAX-WS examples used throughout the course.

2. Build your `WebService`, to trigger the generation of all Java artifacts associated with the `UserManager` service.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Aula Matriz S.A.S.

NetBeans provides a wizard to create a new Web Service using the "contract-first" approach – but only if the project it will belong to is an enterprise project (a web application, or enterprise application). Since you will be working with simple standalone Java projects, to start, you will not be able to take advantage of this wizard.

Task 4 – Implement Web Service Based on Existing WSDL Description

The code generation step produces a Java interface (which will be named after the port type that you specified in your WSDL description) that describes what the API will be that the web service will require from anyone that is used as the implementation of that web service. Now that we have this information available, you can go ahead and implement the web service implementation class.

1. Create a class `labs.UserManagerImpl` which extends `UserManager`.
2. Annotate the class with a `@WebService` annotation. This scenario requires that you add the `endpointInterface` attribute to `@WebService`, to capture the fact that this class is in fact the service implementation for the `UserManager` service described by the WSDL file.
3. Implement all the business operations required. The `User` and `UserDAO` classes that you will need to implement these functions are provided in the `AuctionApp` project:

   NetBeans can help you by providing empty implementations for all of the methods that your `UserManagerImpl` class will need to override.

```
Java Development
→ Java Classes
→ Modifying Java Classes
→ Overriding Methods
```

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Aula Matriz S.A.S.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Task 5 – Deploy Web Service to Standalone JVM

Once the web service is implemented in Java, there's really very little difference between a "code-first" web service and a "contract-first" one. Go ahead and add the logic necessary to deploy your `UserManager` as a web service – you can use the same strategy that you used for the "code-first" implementation of the `ItemManager` web service that you built before:

1. Create a new class, call it `UserManagerRunner`, to be the application that deploys your web service. This new class will have a `main()` function.

2. In `main()`, create an instance of your web service implementation class `UserManagerImpl`, and deploy it as a web service.

   Remember that this involves, at a minimum, two steps:

   (a) Create an instance of the web service implementation class.

   (b) Use the `Endpoint` class to `publish` the instance of the web service implementation class that you created at a particular URL.

   This URL needs to be the full URL that clients will use to contact the web service: it has to include protocol, hostname, port number, and the path that will identify the web service itself. You could use something as simple as:

   ```
   http://localhost:8081/UserManagerService
   ```

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

3. Ensure that the JavaDB database engine is already running.

4. Run this application.

Task 6 – Test Web Service Using SoapUI Plugin

1. Use the SoapUI plugin to generate a test harness for your "contract-first" web service. Check that the web service behaves as advertised.

2. Once you have tested your web services, shut down the web service provider application.

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Aula Matriz S.A.S.

Exercise 4 – Create and Run a Web Service Client

In this exercise, we're going to create and test a simple web service client to the `ItemManager` web service create in Exercise 2.

Task 1 – Define Client Class

1. Create a new project in NetBeans. Call it `WSClients`. Switch to it. This project won't require any dependencies on existing projects – given a WSDL description of the service that the client will use, JAX-WS will generate all the necessary Java artifacts.

   It is convenient to create web service clients in a different project than the one use to create the web service provider for a couple of reasons:

   - The *Separation of Concerns* principle suggests that web service providers and their matching web service clients ought to be maintained separately.
   - The supporting artifacts required by the web service provider and the web service client, or the way in which those artifacts are created, may not be the same. Generating both in the same project could be confusing.

2. Configure the project to support web service calls to the `ItemManager` web service.

   Instead of changing the `build.xml` file to run `wsimport` to generate the necessary Java artifacts explicitly, you will use a wizard in the NetBeans to prepare this project to support web service client logic:

   - Ensure that the `ItemManager` service is up and running.
   - Ask the IDE to configure a web service client reference within the `WSClients` project. When the wizard asks for the WSDL for the target service, enter the URL that can be used to query the service for its own WSDL description:

     ```
     http://localhost:8081/itemManager?WSDL
     ```

     Specify `labs.generated` for the target package for all artifacts.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Aula Matriz S.A.S.

```
Java EE Development
→ Web Services
→ Creating Web Service Clients
```

This step does not actually generate any client logic – it just configures the project so as to support the generation of client logic on demand. It also modifies the project to generate the web service artifacts needed by the client automatically.

Figure 2.6 shows the structure of the `WSClients` project, once the web service client configuration step is done. Note how NetBeans creates a node representing knowledge of the `ItemManager` web service, plus a node for the `labs.generated` package that contains the code for all the Java artifacts required.

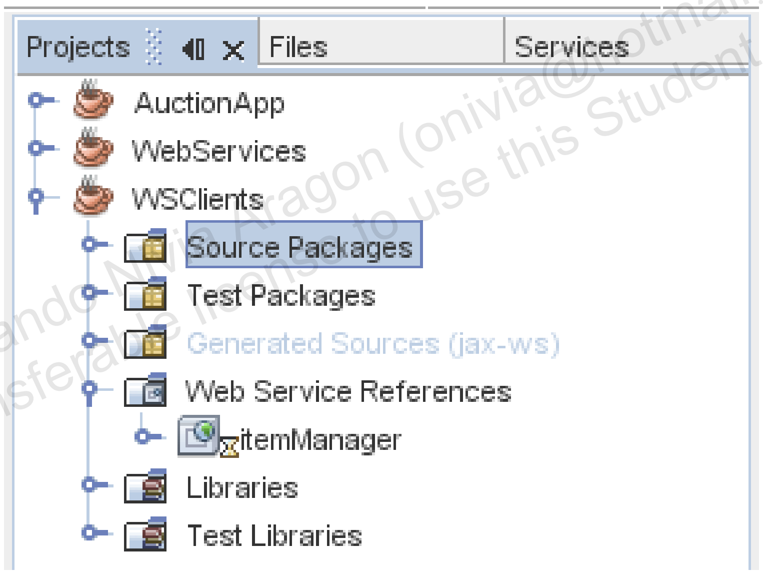


Figure 2.6: NetBeans project structure for `WSClients` project

Once again, NetBeans may refuse to recognize that the artifacts generated by JAX-WS exist. The easiest way to fix this is to add the directory where the generated code is to your project's set of source folders. The folder where these artifacts will appear will be `build/generated-sources/jaxws`.

3. Create a new main class, `clients.ItemManagerClient`.

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Aula Matriz S.A.S.

4. Write logic in your main class to invoke an operation on the web service created in Exercise 2, the `ItemManager` web service.

   The simplest way to do this may be to use the NetBeans wizard to call a web service operation, within the body of the `main()` function of your client application.

```
Java EE Development
→ Web Services
→ Creating Web Service Clients
→ Calling a Web Service Operation
```

Task 2 – Execute Client

1. Run the `ItemManager` web service in `Lab02`, if it is not currently running.
2. Run the `ItemManagerClient` application.
3. Once you have tested your web service, shut down the web service provider application.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Aula Matriz S.A.S.

Lab Summary

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

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Aula Matriz S.A.S.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Aula Matriz S.A.S.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Lab 3

SOAP and WSDL

Objectives

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

- Write WSDL descriptions of web services, using the different styles of SOAP messages that are available.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Aula Matriz S.A.S.

Exercise 1 – Understanding SOAP and WSDL

1. Identify the most popular model and style of SOAP messages from the following list. (Choose one).
   (a) Document literal
   (b) Document encoded
   (c) RPC literal
   (d) RPC encoded
2. State whether the following statements are true or false:
   (a) A WSDL descriptor for a web service is required only for a SOAP-based web service.
   (b) A WSDL file is embedded within a SOAP message.
   (c) WSDL was designed to serve as a standard mechanism for defining the functionality exposed by a web service.
   (d) A web service client can retrieve a service descriptor only from the registry.
3. Identify the primary elements of WSDL from the following list. (Choose one).
   (a) Definitions, types, message, portType, binding, service.
   (b) Types, portTypes, part, input, and output.
   (c) Definitions, part, port, service, and binding.
   (d) Definitions, portType, operations, service, and binding.
4. Identify one or more ways by which the WSDL file of a specific web service is located:
   (a) `http://eshop.com/CustomerServices/POService?WSDL`
   (b) `http://eshop.com/CustomerServices/POService=WSDL`
   (c) `http://eshop.com/CustomerServices/POService/WSDL/OrderService.WSDL`
   (d) `http://eshop.com/CustomerServices/POService?wsdl=Orderservice.wsdl`

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Aula Matriz S.A.S.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

5. State whether the following statements are true or false:

   (a) A WSDL descriptor has an abstract model and a concrete model to describe messages and operations.

   (b) There are only two types of messaging mechanism; one way and Request/Response.

   (c) The binding element specifies the data format and the protocol used in the web service. The standard binding extensions are HTTP, SOAP, and MIME.

   (d) A WSDL file alone cannot be used to generate a web service.

6. Given:

```
<soap:envelope>
  <soap:body>
    <add>
      <x xsi:type="xsd:int">5</x>
      <y xsi:type="xsd:int">5</y>
    </add>
  </soap:body>
</soap:envelope>
```

   What is the style/encoding for this SOAP message? (Choose one).

   (a) RPC/encoded

   (b) RPC/literal

   (c) Document/literal

   (d) Document/literal wrapped

7. Which style specifies that each message consist of a single XML node that corresponds to the operation being invoked, with values encoded as simple text? (Choose one).

   (a) RPC/encoded

   (b) RPC/literal

   (c) Document/literal

   (d) Document/literal wrapped

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Aula Matriz S.A.S.

Exercise 2 – Design a Web Service “Contract-First” Using the RPC/Literal Style

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

In Lab 2, you implemented the `UserManager` web service “contract-first”, by starting with its WSDL description. At the time, you had only seen the Document/literal “wrapped” style of WSDL description – so that is the kind of description that you wrote. In this module, you learned about the different styles of WSDL descriptions that are available.

In this exercise, you will implement that same web service “contract-first”, too – but using the RPC/literal style of WSDL description. Once the WSDL description is ready, you will then implement the Java class that will be the service provider. and then deploy it to a stand-alone Java VM. You can test this service with the SoapUI, too. In the next exercise, you will write a Java client for this service.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Task 1 – Define Description for Web Service in WSDL

1. Create a new project, call it `RPCWebServices`. Switch to it.

   Since you will be running web service implementation classes for the Auction domain out of this project, it will need to have the same dependencies you had specified for the `WebServices` project: the `AuctionApp` project, and the `EclipseLink (JPA 2.0)`, `JAX-WS 2.2`, `JAXB 2.2`, and the Derby (or Java DB) libraries.

2. Create a Java package called `labs`.

```
Java Development
→ Java Packages
→ Creating Java Packages
```

3. Write a second complete WSDL description for the `UserManager` web service, using the RPC/literal style. Since you are working on a new project, you could keep same file name conventions that you used back in Lab 2: name the file `UserManagerSvc.wsdl`, and place it in the `labs` package (folder) you just created.

   You can limit your API to the same functions originally described in Lab 2, Exercise 2.

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Aula Matriz S.A.S.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide

The different styles of WSDL that the author of a description could use share some common structure – after all, they are all attempts to describe the operations offered by web services. Instead of writing this alternative description of the `UserManager` web service from scratch, you could leverage that earlier description: most of the differences between the two styles, RPC/literal and Document/literal "wrapped," can be found in how the two describe the structure (or content) of messages. To reduce the amount of work involved, then, you could:

(a) Copy both files `UserManager.wsdl` and `UserManagerSvc.wsdl` from the `WebServices` project to the `RPCWebServices` project.

   You will not need the `UserManager.xsd` schema file, as long as the operations that make up your service only require Java primitives and Strings for parameters and return types: the Document/literal "wrapped" style requires that you write explicit XML schema definitions for the structure of each message, while RPC/literal only requires schemas for any parameters and return types that are themselves complex Java types.

(b) Edit the `binding` element of the WSDL description, to change it from requiring `"document"` to `"rpc"` style.

(c) Edit the logical description to remove the reference to the included XML schema file.

(d) Edit the logical description so that the `message` elements describe their structure themselves, as is required by the RPC/literal style: using multiple `part` elements within each `message` element.

Task 2 – Customize Project Build Script

In the "contract-first" approach, there are artifacts to be generated from the WSDL definition, and the actual generation has to be presented explicitly – in order to write any code in Java that refers to the service that was initially described by its contract, the developer will need an Java interface for that service. The only way to get that interface is by running the code generator explicitly.

Modify your build script to include the steps require to trigger the generation of artifacts before compilation of the source code for your project – just as you did in Lab 2.

Also, remember to add the `generated/src` folder, where the artifacts generated will be placed, as another source folder within the `RPCWebServices` project; otherwise, NetBeans won't find those artifacts when compiling your own code.

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Aula Matriz S.A.S.

Task 3 – Implement Web Service based on Existing WSDL Description

The code generation step produces a Java interface that describes what the API will be that the web service will require from anyone that is used as the implementation of that web service. Now that you have this information available, go ahead and implement the web service class.

The web service implementation class is defined to implement the operations required by the web service's Java interface. When describing the same service using either RPC/literal and Document/literal styles, the resulting Java interfaces will turn out to be equivalent: the differences in description do not affect the name, parameters, or return type of the operations in the web service – which is all that the Java interface will capture.

You should be able to simply copy the web service implementation class that you wrote in Lab 2 and reuse it here – as long as the descriptions for the two versions of the `UserManager` web service are equivalent.

Task 4 – Deploy Web Service to Standalone JVM

Once the web service is implemented in Java, there's really very little difference between a "code-first" web service and a "contract-first" one. Go ahead and add the logic necessary to deploy your `UserManager` as a web service – you can use the same strategy that you used for the "code-first" implementation of the web service that you built in Lab 2, or the one that you used for the "contract-first" implementation, also in Lab 2.

To avoid confusion, and to be able to run this version of the `UserManager` service along with the earlier one, specify a different URL when you `publish` the web service in your server application.

For example, if the URL parameter to the `publish()` method was:

```
http://localhost:8081/userManager
```

in the `UserManager` server application in Lab 2, you could use:

```
http://localhost:8082/userManagerRPC
```

for the application that will deploy the new version of the `UserManager` web service, built using an RPC/literal description.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Aula Matriz S.A.S.

Task 5 – Test Web Service Using SoapUI Plugin

Use the SoapUI plugin to generate a test harness for your "contract-first" web service. Check that the web service behaves as advertised.

A quick tutorial on how to use `SoapUI` to test a web service can be found in the `resources` folder in your home directory, or directly from the authors' web site: `http://www.soapui.org/gettingstarted/your_first_soapUI_project.html`

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Aula Matriz S.A.S.

Exercise 3 – Clients for "Contract-First" Web Services

In this exercise, you're going to create and test a simple web service client to the new "contract-first" `UserManager` web service defined above.

Task 1 – Define a Client Class

1. Create a new Java Application project, call it `RPCClients`. Switch to it. Remember that client projects need not specify any application-level dependencies – the Java artifacts they need will be generated from the WSDL description to the service they will interact with.

   It would be safest to explicitly associate to your new `RPCClients` project the two libraries `JAX-WS 2.2` and `JAXB 2.2`.

2. Configure this new project to support web service client interactions with the new `UserManager` service.

   Remember that you can do this by configuring the `build.xml` script explicitly to generate the artifacts needed to interact with the target web service, or by telling NetBeans to do so for you by creating a `Web Service Client` reference within the project.

   Remember also that you may have to configure a second source folder for your project, to tell NetBeans where to find the classes that will be created, given the WSDL description for the target service.

3. Explicitly build the `RPCClients` project.

   Normally, NetBeans can compile on demand all the code associated with a Java Application project. Sometimes, though, it misses the step of refreshing the Java artifacts created from the WSDL descriptions of the web services known to the project. An explicit request to build the project takes care of updating these Java artifacts.

4. Create a new main class `clients.UserManagerClient`.

5. Write logic in your main class to interact with the `UserManager` web service created in Exercise 2 above. The simplest way to do this may be to use the NetBeans wizard to generate call a web service operation within `main()`.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Aula Matriz S.A.S.

Task 2 – Test WSDL Clients

Run each of the two `UserManager` clients against their corresponding web service provider applications, to verify they both work:

1. Verify that the JavaDB database server is already running.
2. Run each of the two `UserManager` clients.
3. Once you have tested your web services, shut down the web service provider applications.

Task 3 – Examine Database – Optional

Interacting with your web services could result in the insertion or update of rows into the database tables that represent the persistent entities managed by your web services. You could examine these database tables, to view the changes caused by your web service interactions:

1. Switch to the Services panel in NetBeans.
2. Right-click on the node that represents a connection to the `auction` database, and select `Connect` to establish that connection to the database engine.
3. Expand the connection node, then expand its `Tables` child node.
4. Right-click on the node that represents a table manipulated by one of your web services, then select `View Data`. Figure 3.1 shows what the pop-up dialog might look like.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

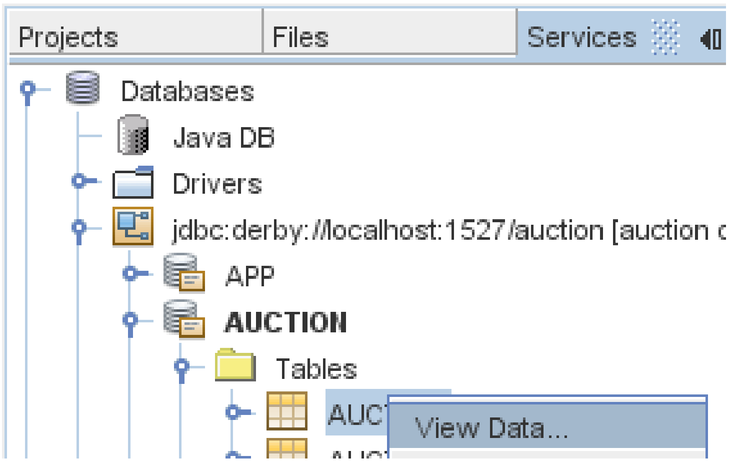


Figure 3.1: View Data Popup Dialog

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Aula Matriz S.A.S.

Lab Summary

Compare the WSDL descriptions for the two variants of the `UserManager` service, the matching Java interfaces generated for each, and the corresponding web service client applications.

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

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Aula Matriz S.A.S.

Lab 4

JAX-WS and JavaEE

Objectives

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

- Deploy POJO web services to a web container.
- Secure POJO web services to a web container, delegating authentication to the container.
- Define a web service in terms of an Enterprise Java Bean.
- Deploy an EJB web service to an EJB container.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Aula Matriz S.A.S.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Exercise 1 – Understanding JAX-WS in Containers

1. Which are considered advantages of deploying POJO web services to a web container? (Choose all that apply).

   (a) HTTP session management built in.

   (b) Scalability and availability features built-in.

   (c) Declarative transaction management.

   (d) Pooling of web service provider instances built in.

2. Which mechanisms are available to define the set of roles that will be available for programmatic authorization within a web service provider class? (Choose all that apply).

   (a) Initialization properties in `web.xml` element corresponding to web service implementation class.

   (b) `@DeclareRoles` annotation on web service implementation class.

   (c) `@RolesAllowed` annotation on web service implementation class.

   (d) `security-constraint` element in `web.xml`.

   (e) `@DeclareRoles` annotation on web service client implementation class.

   (f) `security-role-mapping` elements in `sun-web.xml`.

3. What mechanisms are available for a JAX-WS web service deployed to a web container to obtain the identity of the caller? (Choose all that apply).

   (a) `ServletRequest` as a parameter to the web service method call.

   (b) `ServletRequest` as a resource injected directly in the web service implementation class.

   (c) `SecurityContext` as a resource injected directly in the web service implementation class.

   (d) `ServletRequest` obtained from a `MessageContext`.

   (e) `WebServiceContext` as a resource injected directly in the web service implementation class.

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Aula Matriz S.A.S.

4. Choose the option that best describes how most of the information about a request that is made available to the web service implementation class is provided. (Choose one).

   (a) Directly via dependency injection.

   (b) As properties provided by `WebServiceContext` directly.

   (c) As properties provided by `MessageContext` directly.

   (d) As parameters to the web service method call.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Aula Matriz S.A.S.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Exercise 2 – Deploy POJO Web Services to a Web Container

In Lab 2, you built two simple web services, `ItemManager` (implemented code-first) and `UserManager` (implemented contract-first). They were both designed to be simple POJO web services, and they were both deployed standalone on a JVM.

In this exercise you will deploy them into a web application, so as to leverage the functionality provided by the web container: initially, you will gain the ability to use the automated test harness provided by GlassFish. Later, you will be able to authenticate clients of the web service by delegating the authentication logic to the web container.

POJO web services can be deployed either standalone, using the HTTP server built into Java SE 6, or to any web container. The implementation of the POJO web service itself does not have to change, due to the environment it will be deployed into – any differences at that level are handled by the JAX-WS runtime.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Task 1 – Copy Existing Web Services to Web Application

1. To avoid confusion, create a new web application project (give it the name `POJOsInContainer`) to deploy the two web services into. As before, this new project will need to depend on the `AuctionApp` project and Derby library. (You need not specify either of the JAX-WS 2.2 or JAXB 2.2 libraries, as these are built into GlassFish v3).

   Make sure that you choose to create the project as a `Java EE 6` project.

```
Java EE Development
→ Web Applications
→ Web Application Projects
→ Creating a Web Application Project
```

2. Add to this project two server resource configuration entries:

   (a) Configure a database connection pool, so that this application (and, in particular, the persistence layer that your services rely on) can access the database efficiently. You can name this pool `auctionDbPool`. Use

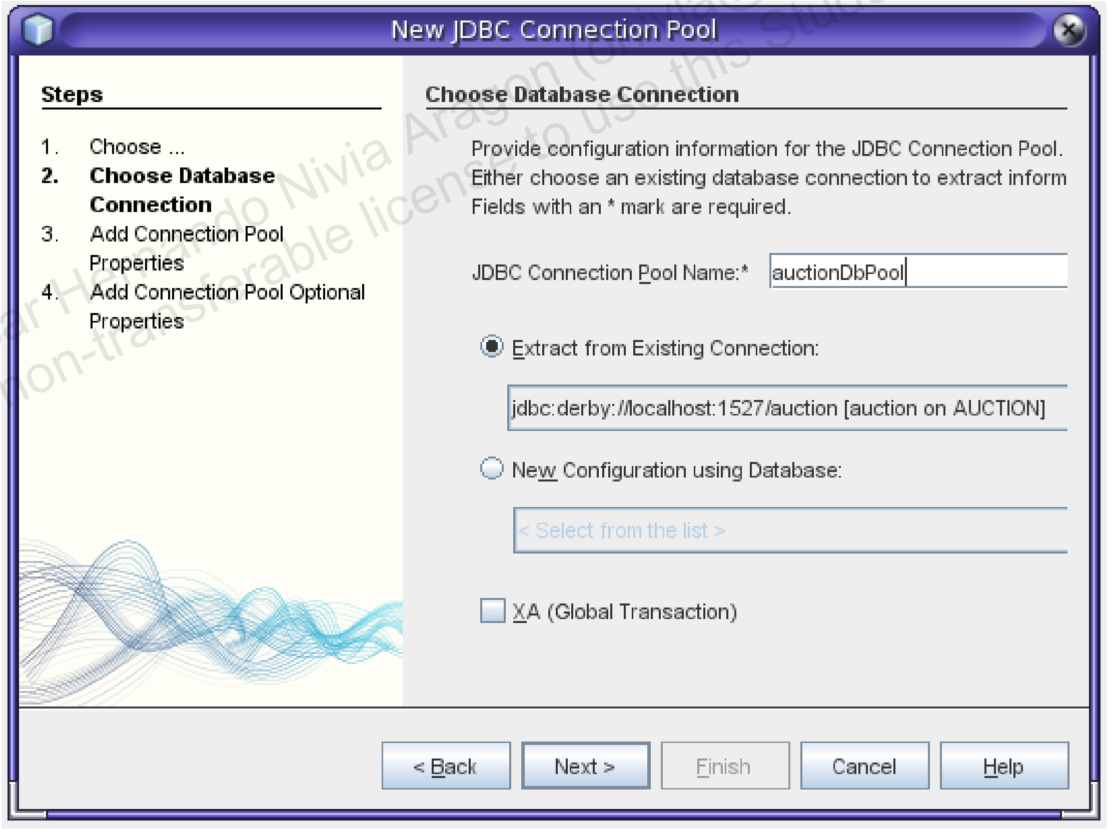
Aula Matriz S.A.S.

the NetBeans connection to the `auction` database that you configured in in the setup lab, so that NetBeans can configure the connection pool based on that same configuration.

(b) Configure a JDBC resource to let the application access the database through the `auctionDbPool` via JNDI lookup. The name of this JDBC resource must be `jdbc/auction`, as that is the resource name embedded in the JPA configuration file for the `AuctionEJBs` project that your project depends on.

Figures 4.1 and 4.2 show what the dialogs to configure a new database connection pool for the GlassFish instance running the application might look like. Figure 4.3 shows the dialog that creates a new JDBC resource.

```
Java EE Development
→ Configuring Java EE Resource
→ Creating a Connection Pool Resource
```


New JDBC Connection Pool
Steps
1. Choose ...
2. Choose Database Connection
3. Add Connection Pool Properties
4. Add Connection Pool Optional Properties
Choose Database Connection
< Back
Next >
Finish
Cancel
Help

Provide configuration information for the JDBC Connection Pool.
Either choose an existing database connection to extract inform
Fields with an * mark are required.

JDBC Connection Pool Name:* auctionDbPool

(●) Extract from Existing Connection:
jdbc:derby://localhost:1527/auction [auction on AUCTION]

( ) New Configuration using Database:
< Select from the list >

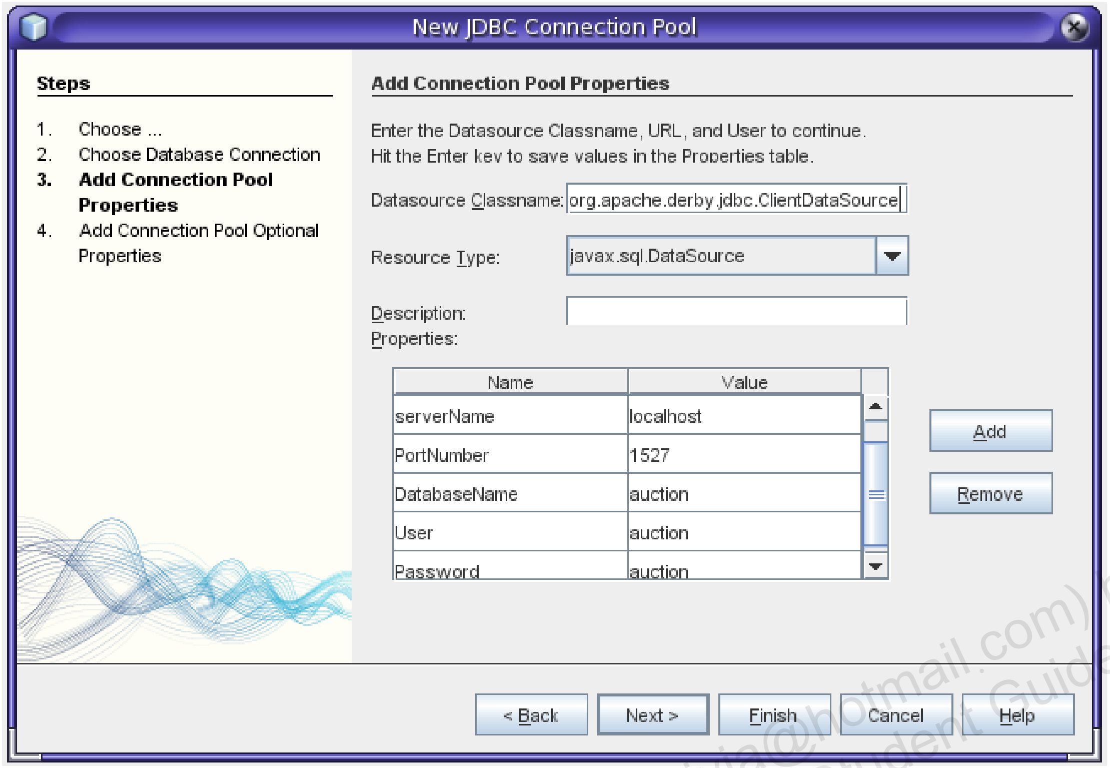
☐ XA (Global Transaction)

Figure 4.1: Entering New Connection Pool Name

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.
Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Aula Matriz S.A.S.


New JDBC Connection Pool
Steps
1. Choose ...
2. Choose Database Connection
3. Add Connection Pool Properties
4. Add Connection Pool Optional Properties
Add Connection Pool Properties
Add
Remove
< Back
Next >
Finish
Cancel
Help

Enter the Datasource Classname, URL, and User to continue.
Hit the Enter key to save values in the Properties table.

Datasource Classname: org.apache.derby.jdbc.ClientDataSource

Resource Type: javax.sql.DataSource

Description:

Properties:

Name	Value
serverName	localhost
PortNumber	1527
DatabaseName	auction
User	auction
Password	auction

Figure 4.2: Entering JDBC Information for new Connection Pool

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

```
Java EE Development
→ Configuring Java EE Resource
→ Creating a JDBC Resource
```

3. Copy the code for the two services `ItemManager` and `UserManager` from the `WebServices` project you created in Lab 2 into this new project.

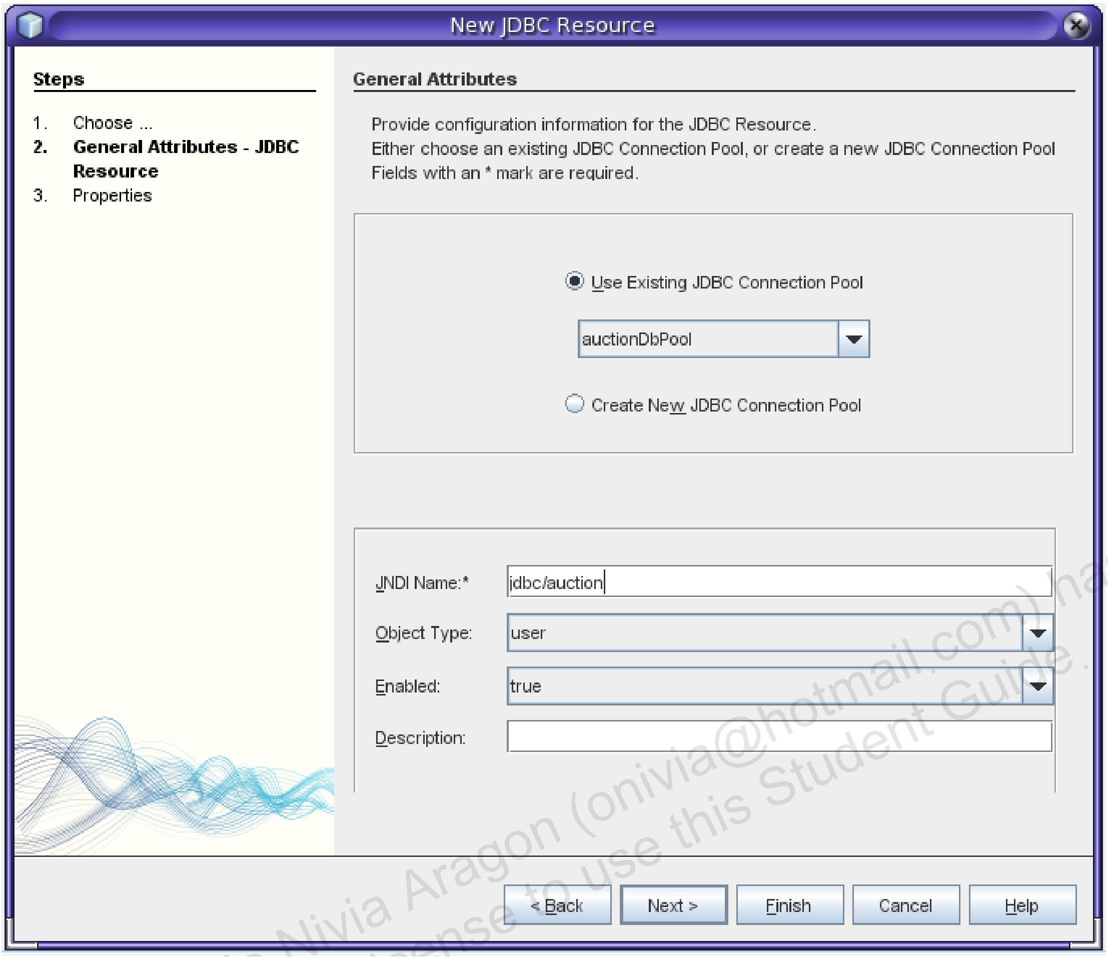
   To avoid having to make changes to the `build.xml` script in this new project, you could use NetBeans wizards to help you introduce these web services to this web application:

   (a) Use the NetBeans "create new web service" wizard to create a new empty `ItemManager` web service "code-first" (from scratch). Place it in the `labs` package.

   (b) Once the empty web service implementation class is present, copy the implementation of that class from the `ItemManager` class in the `WebServices` project.

   (c) Copy the WSDL description for the `UserManager` web service from the

Aula Matriz S.A.S.



Provide configuration information for the JDBC Resource.
Either choose an existing JDBC Connection Pool, or create a new JDBC Connection Pool
Fields with an * mark are required.

(●) Use Existing JDBC Connection Pool: auctionDbPool
( ) Create New JDBC Connection Pool

JNDI Name:* jdbc/auction
Object Type: user
Enabled: true
Description:

Figure 4.3: Creating JDBC Resource

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

`WebServices` project into the `labs` folder of `POJOsInContainer`.

(d) Use the NetBeans "create new web service" wizard again, to create a new web service from a given WSDL description. Have the wizard look at the WSDL description for `UserManager` in this project.

The NetBeans wizard will ask you for a web service name. This is the name that it will use for the class that it will create as an implementation of the web service described by the WSDL file you specify. For consistency, name it `UserManagerImpl`.

(e) Copy the definitions of the members of the `UserManagerImpl` class from the `WebServices` project to this one.

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Aula Matriz S.A.S.

```
Java EE Development
→ Web Services
→ JAX-WS Web Services
→ Creating an Empty JAX-WS Web Service
```

```
Java EE Development
→ Web Services
→ JAX-WS Web Services
→ Creating a JAX-WS Web Service From a WSDL File
```

4. No explicit configuration changes to the web application (that is, changes to the `web.xml` file) are needed – JAX-WS will configure URLs to access the web services within the application by default, based on their class name. The URLs to access services will have the form:

    http://*host*:*port*/*context*/*serviceName*

   where *serviceName* by default will be the class name for the POJO web service, with the suffix `Service`. However, it is possible to control the service name component of that URL: the `serviceName` attribute of the `@WebService` annotation allows the developer to specify the name clients will use to contact the web service in question. Change the definition of the `ItemManager` web service so that clients can contact it at the URL:

```
http://localhost:8080/POJOsInContainer/ItemManagerWS
```

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Task 2 – Deploy and Test Services in Web Application

Deploying the `POJOsInContainer` web application to a web container will automatically deploy the web services contained within it.

A benefit of deploying JAX-WS services to a web container is that an automated test harness for those web services is provided for free. The `ItemManager` web service is available at the URL:

```
http://localhost:8080/POJOsInContainer/ItemManagerWS
```

To see this test harness in action:

1. Deploy the `POJOsInContainer` web application.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Aula Matriz S.A.S.

2. Open up a browser, and visit the URL:
`http://localhost:8080/POJOsInContainer/ItemManagerWS?Tester`

```
Java EE Development
→ Enterprise Application Projects
→ Deploying Java EE Applications
```

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Aula Matriz S.A.S.

Exercise 3 – Secure POJO Web Services in Web Container

Earlier, you deployed the two web services `ItemManager` and `UserManager` to a web container, as part of a web application. One of the advantages of doing this is that it becomes possible to secure those web services, delegating authentication to the web container. In this exercise, you will secure access to the `UserManager`, so that only administrators can create new users.

Task 1 – Define Security Constraints on Web Service

You have to capture, as part of the description of the service, that there are two kinds of users of the `UserManager` web service, "administrators" and everyone else, and that only administrators are allowed to add new users to the application:

1. For convenience, let's define two *roles*, or groups of users, called `user` and `administrator`. The set of roles expected by a JAX-WS web service can be captured by annotating the web service implementation class with a `@DeclareRoles` annotation, listing those roles by name.

2. A `@RolesAllowed` annotation can be specified at class level, to specify a default requirements that the caller belong to certain roles to call any operation on the web service implemented by that class. It can also be specified at method level, to restrict access to a particular operation in that class. Capture the constraints that callers in the `user` role can call any of the operations in either `ItemManager` or `UserManager`, but only callers in the `administrator` role can call the `addUser` operation in `UserManager`.

You could also log the name of the person adding a new user to the container's log file, just to show that authenticated user information is provided to the web service.

Task 2 – Define Security Constraints on Web Application

Capturing authentication constraints at the level of the web service implementation classes isn't enough. JAX-WS will delegate authentication to the web container – but the web container requires additional configuration information, in order to perform authentication:

1. Create an empty `web.xml` configuration file, if it doesn't exist, yet.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Aula Matriz S.A.S.

As of JavaEE6, most of the configuration for web applications can be achieved via annotations, so `web.xml` files may not exist – until the web application needs to capture something about the configuration of the application that has no (convenient) annotation associated with it.

```
Java EE Development
→ Web Applications
→ Web Deployment Descriptions
→ Creating the Standard Deployment Description
```

2. A `login-config` element has to be added to the web application's configuration file, `web.xml`. This element is used to configure the authentication mechanism that the web container will use. Set it to `BASIC`, which expects clients to provide userid and password information in an HTTP header. Set the realm name to `file` – which is the simplest realm supported by GlassFish.

```
Java EE Development
→ Web Applications
→ Web Deployment Descriptors
→ Security Configuration
```

3. Add `security-role` elements for the two roles needed in this exercise, `user` and `administrator`.

4. A `security-constraint` element has to be added to `web.xml`, too. Label it `AuctionServices`.

   This element indicates which URLs are to require user authentication, for a given HTTP method, and which roles a caller has to belong to, in order to be allowed access to the resource. Set the HTTP method to `POST` only – clients will need to request the WSDL description of the service from it, and this is done via a `GET` request.

   The security constraint element will specify the set of URLs to be protected, and the set of user roles that should be allowed access to these resources:

   - You should protect both web services, so you should specify both URLs within the security constraint element.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Aula Matriz S.A.S.

- The goal is to allow either callers in either `user` or `administrator` roles to call any of the Auction-related services – but to only allow callers in the `administrator` role to call the `addUser` operation.

  The web container can only enforce access on the basis of the incoming URL – the particulars of the web service call intended by each request are not visible, yet. As a result, the `web.xml` security constraint can only require that callers be in either role to refer to either web service at all – the further constraint that only administrators can call `addUser` will have to be enforced on the web service side.

5. The set of users expected by the application, along with the roles that each user will belong to, must be specified, too - but this is not part of the product-independent web application configuration.

   In the case of GlassFish in particular, the product-specific configuration file is `sun-web.xml`. Add `security-role-mapping` elements to this file, for each of the users that will require authentication to use the application.

   The lab instructions assumes there will be two users, with userids `tracy` and `kelly`; their passwords will be set to `password` in a later step. `tracy` is assumed to be an `administrator`, while `kelly` is a regular `user`.

   The `sun-web.xml` can be found in the `Configuration Files` node within the `POJOsinContainer` project structure, as show in Figure 4.4. NetBeans supports both a raw XML and a GUI-driven options for the editor for this file. Figure 4.5 shows what the GUI-driven editor for `sun-web.xml` looks like.

The set of roles that can be specified in the web application's `sun-web.xml` is *application-specific* – different web applications deployed to the same web container could each have different roles defined for each application.

For convenience, `sun-web.xml` can also map *system-wide* roles to application-specific ones – this would allow for the administration of a number of web applications, without requiring a complete of all authorized used separately for each individual application: the system-wide assignment of system roles to principals would determine their application-specific role assignments within each web application.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Aula Matriz S.A.S.

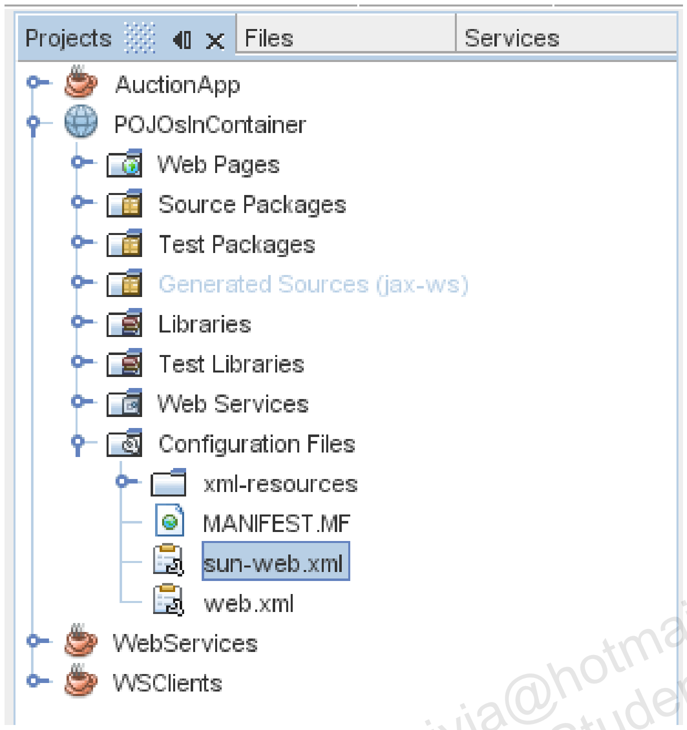
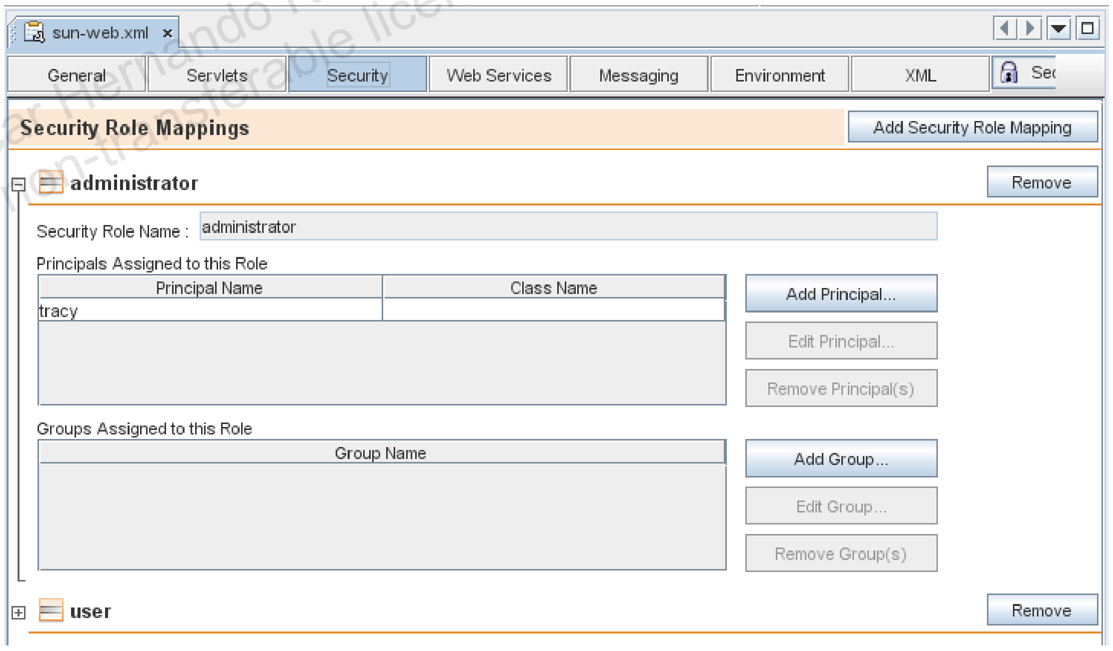


Figure 4.4: Accessing `sun-web.xml` File



Security Role Name : administrator

Principals Assigned to this Role

Principal Name	Class Name
tracy	

Groups Assigned to this Role

Group Name

Figure 4.5: `sun-web.xml` GUI-driven Editor

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Aula Matriz S.A.S.

Task 3 – Configure Users in Authentication Realm

The actual authentication of principals is outside the scope of the specification of a web application – it is a responsibility of the web container. Therefore, configuration of the information associated with known principals (for example, their user names and passwords) must be done at the level of the web container.

In the case of GlassFish, the ability to define principals is available through the GlassFish *administration console*. This administration console is a web application deployed within GlassFish, available at the url:

```
http://localhost:4848
```

To access it, you may need the GlassFish administrator's login (which by default is `admin`) and password (by default `adminadmin`). You may have to start the server, if it is not yet running, before you can access its administration console.

```
Server Resources
→ Java EE Application Servers
→ Administering Security
→ Adding a File Realm User
```

You can also access the GlassFish administration console through NetBeans: right-click on the node for `GlassFish v3 Domain`, in the `Services` tab, as shown in Figure 4.6.

Once on the administration console's home page, the security configuration page can be found under `Configuration/Security` in the navigation bar on the left. Configuration of the default file realm (specified in that main security configuration page) can be found under `Configuration/Security/Realms/file`. You can click on the `Manage Users` button there, to add users and their credentials to the realm.

The lab instructions assumes there will be two users, with userids `tracy` and `kelly`. Both of them should have their passwords set to `password`.

Task 4 – Deploy Service in Web Application

Since the two web services are now packaged up as part of a web application, deploying that web application will automatically deploy the two web services along. Go ahead and deploy the `POJOsinContainer` web application.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Aula Matriz S.A.S.

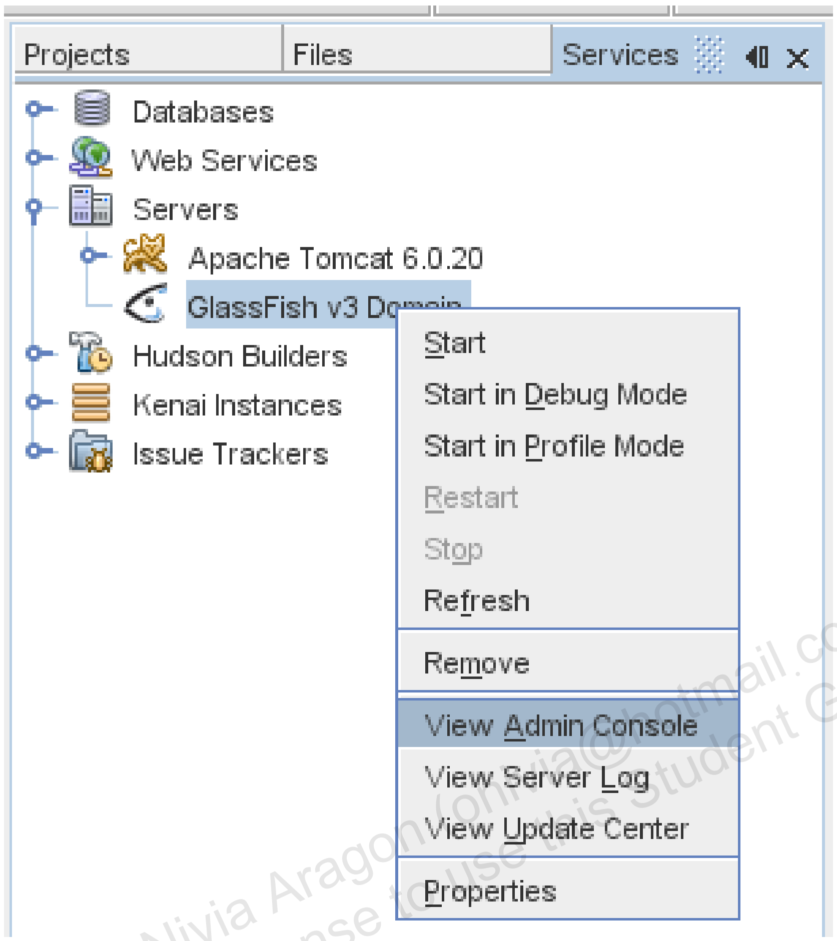


Figure 4.6: NetBeans Link To GlassFish Admin Console

Deploying this application should start the database server running automatically, if it was not yet running. Ensure that the database server is running, when the application is deployed.

Normally, you could use the test harness provided in GlassFish to test web services – except that the test harness is not sophisticated enough to pass authentication information as part of the test request. To test the secured web services, you will have to write a JAX-WS client application that can authenticate itself when it calls out to the web services.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Aula Matriz S.A.S.

Exercise 4 – Creating an Authenticating Client

In this exercise, you're going to create and test a simple authenticating web service to the secured `UserManager` web service defined above.

Task 1 – Define a Client Class

1. Create a new Java Application project, call it `SecuredClient`. Switch to it. Remember that client projects need not specify any application-level dependencies.

To work around a bug in JAX-WS, JAXB, and GlassFish, please add the following XML element to the `build.xml` script for your project:

```
<taskdef name="wsimport"
         classname="com.sun.tools.ws.ant.WsImport"/>
```

Place this line after the `import` element in that file.

2. Configure this new project to support web service client interactions with the new secured `UserManager` service.

   You deployed this secured service within a web application; its URL will therefore be relative to that web application. If you followed the naming conventions mentioned earlier, the URL to your secured service should be:
   `http://localhost:8080/POJOsInContainer/UserManagerSvc`

   It would be easiest to simply let NetBeans configure your project for you, through the `New Web Service Client` wizard.

   Remember that to query a web service for its WSDL description, you have to append the suffix `?WSDL` to its service URL.

   Remember also that you may have to configure a second source folder for your project, to tell NetBeans where to find the classes that will be created, given the WSDL description for the target service.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Aula Matriz S.A.S.

The NetBeans wizard will ask you what package you would like to place the generated artifacts into. You **must** leave this blank! There seems to be a bug in the way JAX-WS and JAXB are handling XML namespaces – and the workaround is to let them default the artifacts' package name to match the namespace name.

3. Create a new main class `labs.UserManagerClient`.
4. Write logic in your main class to invoke the `addUser` operation on your secured web service. The simplest way to do this may be to use the NetBeans wizard to generate call a web service operation within `main()`.

   There is one catch, however. You will also need to provide authentication information for the client proxy to send as part of the web service request, so that the server can validate whether you should be allowed to execute that operation.

   You provide authentication data to a JAX-WS service via the interface `BindingProvider` that the port proxy implements: it can provide you with a request context, which is a map into which you can store properties for the username and password to provide to the server. Code 4.9 in your student guide is a code example similar to the code you will need to write here.

Task 2 – Test WSDL Clients

Run the `UserManager` client against its corresponding web service provider application, to verify it works.

Task 3 – Shut Down Web Services

Undeploy the web service provider application(s) you deployed in this exercise, and shut down the application server.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Aula Matriz S.A.S.

Exercise 5 – Define and Deploy EJB-based Web Services

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

The web services you have built so far use data access objects (DAOs) to manipulate persistent domain objects. If the task they need to perform becomes complex enough, the web services themselves may need to manage the transactions used to perform the work required by each operation – and this could be complex logic to write and maintain.

In this exercise, you will modify your web services so they are implemented as Enterprise java Beans, rather than POJOs. This will allow you to delegate the implementation of the transaction management that would be required to the EJB container – though you will still to specify (declaratively) what the transaction semantics required by your operations will be.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Task 1 – Copy Existing Web Services to Enterprise Application

As a new feature in JavaEE6, an enterprise application can be packaged up and deployed as a simple web application – an EJB container, and many of the services that enterprise applications rely on, are available alongside the JavaEE6 web container. In this lab, you will take advantage of this new feature.

1. Copy your existing `POJOsInContainer` web application project into a new project, called `EJBWebServices`. This will avoid confusion between the new EJB-based services you are about to write, and the services deployed earlier as POJO web services in a web container.

   Copying a project in NetBeans is straightforward: in the `Project` view, right-click on the project you want to copy, to bring up the context menu, then choose `Copy...`. A dialog will ask you for the destination folder and name for the new project: place it in the same folder as `POJOsInContainer`, and name it `EJBWebServices`.

   However, you are going to modify your web services so that they are implemented as Enterprise JavaBeans. At the same time, the DAOs that the web service providers rely on will also become EJBs – and so you will benefit from the ability to manage transactions contexts transparently that is built into the EJB framework.

   You will achieve the switch from POJO DAOs to EJB DAOs by changing your project dependencies: instead of depending on the `AuctionApp` project, as you have been doing so far, you will now make this new project

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Aula Matriz S.A.S.

depend on the `AuctionEJBs` project (which is inside the same folder as the original `AuctionApp`).

Until you modify your web service implementation classes, as described in the next step, NetBeans will report problems with those classes, after you update your project's dependencies. This is due to two small differences:
- The APIs are a little different, the POJO DAOs and the EJB DAOs.
- The package where the DAOs are placed is different.

So the errors are to be expected, until you fix your code in the next step.

Task 2 – Rewrite Existing Web Services as EJBs

Modifying the web service implementation classes so that they are implemented as EJBs is straightforward: since the two specifications (JAX-WS and EJB) can be said to be orthogonal to (that is, independent of) each other, you only need to add the specification that the class is to be an (stateless) EJB to the existing code – and fix the problems introduced by the change in dependent project.

The one important difference between POJOs and EJBs to keep in mind is that it is the responsibility of the application to instantiate POJOs, while it is the responsibility of the container to instantiate EJBs: you must modify your web service implementation classes so that, where your POJO web service implementation class used to instantiate a POJO DAO explicitly, you will now ask the container to inject an EJB DAO into your EJB-based web service implementation class.

1. Specify that the web service implementation classes are to be implemented as EJBs through use of the `@Stateless` annotation.

2. The motivation behind the choice to rewrite your web services as EJBs was to delegate transaction management to the EJB container. For this to work, however, both the web service implementation classes themselves, and the helper objects they delegate onto, must be EJBs.

   The `AuctionEJBs` project exposes the persistent doman types needed, along with EJB-based DAOs – but you must modify your code to match those new EJB-based DAOs:

   - The `null` first argument to all DAO calls is no longer needed.

   - The package name changes to `Model.dao.ejbs`.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.
Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Aula Matriz S.A.S.

- Helper EJBs are *injected* with `@EJB` annotations, not instantiated explicitly.

Update your web services implementation classes to fix these issues.

Task 3 – Deploy EJb-Based Services in Web Application

Since the two EJB-based web services are also packaged up as part of a web application, deploying that web application will automatically deploy the two web services along. Go ahead and deploy the `EJBWebServices` web application.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Aula Matriz S.A.S.

Exercise 6 – Creating a Client for an EJB-Based Web Service

Task 1 – Create and Run a Client Application

Since the EJB-based web services you just created were based on the access-restricted POJO web services from an earlier lab, they too require authentication data from their clients: an authenticating client will be required, to test the EJB-based web services:

1. The simplest way to build an authenticating client for the new EJB-based web services is to copy the authenticating client you build in Exercise 4. However, to avoid confusion (and perhaps interference between clients), you should copy the complete `SecureClient` project into a new project, call it `EJBSecuredClient`.

2. Remove the reference to the `UserManagerSvc` currently configured in the `EJBSecuredClient` project. This is a reference to the web service in the `POJOsInContainer` service, which is not your intended target for this new client.

3. Add a web service client reference to the EJB-based `UserManagerSvc` in the `EJBWebServices` project. If you followed the naming suggestions before, the URL to this service should be:

   ```
   http://localhost:8080/EJBWebServices/UserManagerSvc
   ```

   Remember that to query a web service for its WSDL description, you have to append the suffix `?WSDL` to its service URL.

4. The existing `UserManagerClient` application should work with no changes, as a client for the new web service. Edit that class, if necessary, then execute it to test your EJB-based web service.

Task 2 – Shut Down Web Services

Undeploy the web service provider application(s) you deployed in this exercise, and shut down the application server.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Oscar Hernando Nivia Aragon (ornivia@hotmail.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Aula Matriz S.A.S.

Lab Summary

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

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Aula Matriz S.A.S.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Lab 5

Implementing More Complex Services Using JAX-WS

Objectives

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

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

- Apply JAXB to pass complex objects to and from a web service.
- Understand how to map Java exceptions from a web service provider to SOAP faults.
- Inject attributes into JAX-WS web service endpoints.

Aula Matriz S.A.S.

Exercise 1 – Understanding JAX-WS

Given:

```
<message name="sayHello">
  <part name="parameters" element="tns:sayHello"/>
</message>
<message name="sayHelloResponse">
  <part name="parameters" element="tns:sayHelloResponse"/>
</message>
<message name="UDException">
  <part name="fault" element="tns:UDException"/>
</message>
<portType name="UserDefinedExceptionWS">
  <operation name="sayHello">
    <input message="tns:sayHello"/>
    <output message="tns:sayHelloResponse"/>
    <fault name="UDException" message="tns:UDException"/>
  </operation>
</portType>
```

1. Which is a valid endpoint signature for this WSDL fragment? (Choose one)

   (a) `public void sayHello() throws UDException`

   (b) `public String sayHello(String name) throws UDException`

   (c) `public String sayHello() throws UDException`

   (d) `public void sayHello(String name) throws UDException`

2. When do service exceptions occur? (Choose one)

   (a) When a web service call does not result in a fault.

   (b) When a web service call results in the service returning a fault.

   (c) Whenever a web service call occurs.

   (d) Whenever a web service call does not occur

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Aula Matriz S.A.S.

3. Which parameter(s) is/are required to configure the constructor for `javax.xml.ws.soap.SOAPFaultException` ?

   (a) fault

   (b) fault, actor

   (c) fault, actor, code

   (d) fault, actor, code, node

4. Which of the following technologies uses `SOAPFaultException` to wrap and manage a SOAP-specific representation of faults?

   (a) DOM

   (b) SAX

   (c) SAAJ

   (d) XML

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Aula Matriz S.A.S.

Exercise 2 – Pass Complex Types

The services you have implemented so far, `ItemManager` and `UserManager`, have been limited by the fact that you had only seen how to pass simple types. In this Module, you learned how to pass complex types, so you can make these two services more complete.

You should continue to work on the `EJBWebServices` project (where you have written EJB-based web services): extending the EJB-based web services is more straightforward, since your web services can delegate transaction management and security access control to the EJB infrastructure.

Task 1 – Complete the API of the ItemManager Service

1. Complete the APIs of the `ItemManager` web services, to include operations that require complex types as input or output parameters:
   (a) Add a `findItem` operation to your `ItemManager`, which accepts an id as a parameter, and returns the `Item` with that id.
   (b) Add a `updateItem` operation to your `ItemManager`, which accepts an `Item` as a parameter, to update the persistent version of that item.

```
Java EE Development
→ Web Services
→ JAX-WS Web Services
→ Adding Operations to a JAX-WS Web Service
```

2. Deploy the updated web service.

Task 2 – Update ItemManager Web Service Client

In earlier labs, you wrote standalone Java client applications to test the web services you built then – and in particular, you wrote a standalone client application to test the `ItemManager` web service. In this lab, you have updated that service to add new functionality. To test it, you will extend that earlier client to invoke the new operations you just added to `ItemManager`:

1. NetBeans projects which include web service client logic have a reference to the target web service as part of the configuration of those projects.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Aula Matriz S.A.S.

This reference is used to generate the client-side Java artifacts that will be needed to call on that service – but, if the web service changes, the reference to the web service in the client project must be updated.

Refresh that web service client reference, to update the Java artifacts available to the client application. You may also have to rebuild the project explicitly, to ensure that those Java artifacts are regenerated and recompiled.

```
Java EE Development
→ Web Services
→ Creating Web Service Clients
→ Refreshing Web Services and Clients
```

2. Add logic to the client application to invoke the new operations available.

   Alternatively, you could copy the existing client application, to create a new application to invoke the new operations.

Task 3 – Complete the API of the UserManager Service

1. Complete the WSDL description of the `UserManager` web services, to include operations that require complex types as input or output parameters:

   (a) Add a `findUser` operation to your `UserManager`, which accepts an id as a parameter, and returns the `User` with that id.

   (b) Add a `updateUser` operation to your `UserManager`, which accepts an `User` as a parameter, to update the persistent version of that user.

   Writing the additional XML schema types required by the need to pass `User` objects back and forth between client and server sides could be cumbersome – but you can use the WSDL descriptions for the `ItemManager` from the previous task to help you modify the `UserManager` WSDL:

   - New operations in the `portType` element.
   - New bindings for the operations in the `binding` element.
   - New `message` elements, for the input and output messages for the new operations.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Aula Matriz S.A.S.

- New XML schema types that describe the structure of the new messages.

Since authoring WSDL descriptions can be cumbersome, you will find copies of the completed WSDL descriptions in the `resources` folder in your home directory, as well as a completed project in the `solutions` folder there.

2. Refresh the Java artifacts generated on the server side from your WSDL description. This will add new methods in the web service endpoint interface corresponding to the new operations you just added to the WSDL description of the web service.

```
Java EE Development
→ Web Services
→ JAX-WS Web Services
→ Refreshing a JAX-WS Web Service
```

NetBeans caches a copy of the WSDL description of a "contract-first" web service – so it is probably a good idea to allow the wizard to replace that cached copy.

Also, you could let the NetBeans wizard create a new version of the implementation class for you. It will save the old one under a different name, so you can copy you existing code from the old version to the new one. Alternatively, you could continue to use the e existing implementation class: since you implement the service endpoint interface, NetBeans would be able to help you complete the implementation of your class by providing empty methods to match the new operations in the updated interface.

```
Java Development
→ Java Classes
→ Modifying Java Classes
→ Overriding Methods
```

3. Add implementations of the new methods to the web service implementation class that you wrote, to actually provide the services included in the new WSDL description for the `UserManager` web service.

4. Deploy the updated web service.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Aula Matriz S.A.S.

Task 4 – Update UserManager Web Service Client

Earlier, you wrote a standalone client application to test the `UserManager` web service. In this lab, you have updated that service to add new functionality. To test it, you will extend that earlier client to invoke the new operations you just added to the service:

1. Refresh the web service reference to the `UserManager` service in your client project, to update the Java artifacts available to the client application in that project.

2. Update the client application to invoke the new operations available.

3. Run the updated client application, to test that your new web services are operating correctly.

Task 5 – Shut Down Web Services

Undeploy the web service provider application(s) you deployed in this exercise, and shut down the application server.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Aula Matriz S.A.S.

Exercise 3 – Building More Complex Web Services Using JAX-WS

The web services you are implementing are there to support an Auction application. In this exercise, you will write some of the more higher-level services associated with such an application, using the code-first approach.

You will implement an `AuctionManager` web service. Some of the Auction operations require the `User` who is making the request (the seller, when creating an auction, or the bidder, when placing bids). You could implement the `AuctionManager` web service in your original `WebServices` project as another standalone web service – but your service could not verify the identity of the caller. It would be more realistic to build this new web service on either the `POJOsInContainer` or the `EJBWebServices` projects, so as to benefit from the ability to authenticate users offered to POJO web services, or to EJB-based web services. You will continue to work in the `EJBWebServices` project.

Task 1 – Design and Implement a AuctionManager Web Service

An Auction application must support, at a minimum, the ability for users to create and look up auctions, and place bids on existing auctions. Figure 5.1 could be a UML diagram for the operations that the `AuctionManager` would offer.

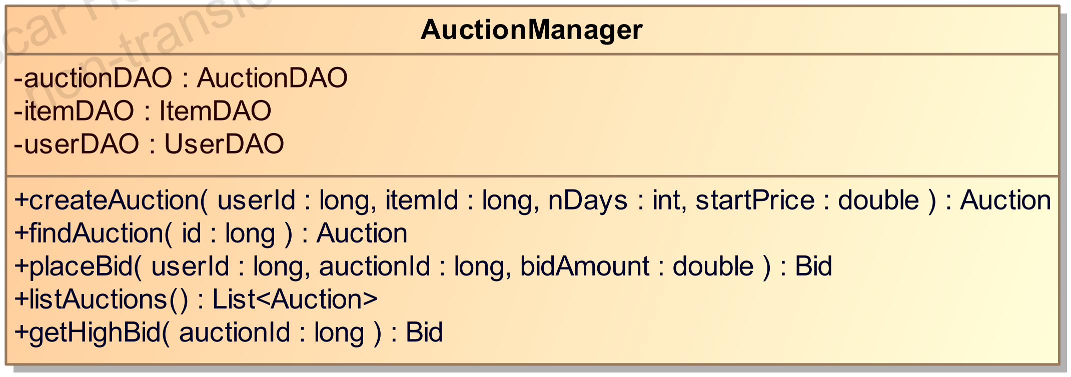

AuctionManager
-auctionDAO : AuctionDAO
-itemDAO : ItemDAO
-userDAO : UserDAO
+createAuction(userId : long, itemId : long, nDays : int, startPrice : double) : Auction
+findAuction(id : long) : Auction
+placeBid(userId : long, auctionId : long, bidAmount : double) : Bid
+listAuctions() : List<Auction>
+getHighBid(auctionId : long) : Bid

Figure 5.1: Possible UML Diagram for AuctionManager Web Service

1. Define a `AuctionManager` Web Service as the class `labs.AuctionManager` in your web service project.
2. Include the operations listed in Figure 5.1 above.

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.
Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Aula Matriz S.A.S.

The AuctionManager web service is a bit more complex than the services you have built before:

- Some of the operations provided (createAuction(), findAuction(), or listAuctions()) can be implemented in terms of the appropriate DAO, in this case the AuctionDAO.
- Some of the operations (placeBid(), getHighBid() must be implemented as calls to business methods of the proper Auction instance (which can be retrieved using the AuctionDAO).
- Some of the parameters to server-side methods (like the seller and item in the call to createAuction() require business objects - but the caller only provides the ids for those objects as web service parameters: the AuctionManager will need to call on an ItemDAO and an UserDAO to obtain the appropriate instances to use.

3. Add the ability for the new web service to authenticate users:

   (a) Declare that all may call any operations on AuctionManager, using annotations on the Java class.

   Remember that the @PermitAll annotation captures this constraint.

   (b) Require that the container obtain identity information for all callers of the AuctionManager service.

   This involves modifying the web.xml configuration file for the web application: the URL used to contact the AuctionManager web service must be added to the list of protected resources that require container-level authentication. Remember to limit authenticated access to this URL to POST requests, only – it may be better if the request to obtain the web service's description directly from the web service is always allowed (and this request is issued as a GET request).

4. Deploy the updated web service project.

Task 2 – Create an AuctionManager Web Service Client

Since the AuctionManager web service is another authenticating web service, you will need to create another Java client application to interact with it:

1. Create a new client application to test your new AuctionManager web service in the EJBSecuredClient project.
2. Run your new client application, to test the AuctionManager web service.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Aula Matriz S.A.S.

Task 3 – Shut Down Web Services

Undeploy the web service provider application(s) you deployed in this exercise, and shut down the application server.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Aula Matriz S.A.S.

Lab Summary

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

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Aula Matriz S.A.S.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Aula Matriz S.A.S.

Lab 6

JAX-WS Web Service Clients

Objectives

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

- Understand how to create web service clients using JAX-WS.
- Understand how to create web service clients using JAX-WS that support asynchronous pull-based interactions.

Oscar Hernando Nivia Aragon (olivia@hotmail.com) has a non-transferable license to use this Student Guide.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Aula Matriz S.A.S.

Exercise 1 – Understanding How to Create Web Service Clients

1. Which is the proper annotation to inject a `Service` instance into a JavaEE web service client? (Choose one).
   (a) `@WebService`
   (b) `@WebServiceRef`
   (c) `@Resource`
   (d) `@EJB`

2. Which is the proper way to enable schema validation on a JAX-WS client? (Choose one)
   (a) Add a binding declaration to the WSDL description of the service.
   (b) Add the `@SchemaValidation` annotation to the web service client implementation class.
   (c) Create an instance of the proper `WebServiceFeature` subclass, then pass it to the `getPort()` call on a `Service`.
   (d) There is nothing to do: schema validation is enabled by default on JAX-WS web service clients.

3. Which is the proper way to enable support for asynchronous interactions in a JAX-WS web service port proxy? (Choose one).
   (a) Add the `@Asynchronous` annotation to the method to invoke asynchronously on the web service implementation class.
   (b) Add the `@Asynchronous` annotation to the field that will hold the reference to the JAX-WS port proxy on the client side.
   (c) Add a binding customization to the WSDL description of the web service, on the client side.
   (d) One cannot invoke asynchronous operations at the level of the JAX-WS proxy type; the only way to invoke an operation asynchronously in JAX-WS involves bypassing the proxy and using the dynamic JAX-WS `Dispatch` API.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Aula Matriz S.A.S.

Exercise 2 – Building an Asynchronous Web Service Client

Task 1 – Creating an Asynchronous Web Service Client

All your client applications so far have assumed that there was nothing else going on at the client, at the time that it issued a call for an operation on one of your web services – so it was reasonable for the client to simply issue the call, then wait for the answer to come back.

One could imagine a more sophisticated Auction client that presents its user with status information on multiple auctions concurrently, or one that updates the status displayed for the auction that the user is currently interested in, live, even in the middle of performing other operations - like bidding on that same auction.

In this exercise, you will write a client for the `AuctionManager` web service that issues a request to place a bid on an auction, without blocking while the request to place that bid is processed on the server side. This is the kind of more sophisticated logic that could be required for the two scenarios presented above.

1. Deploy the `EJBWebServices` project once again (you should have undeployed it at the end of the previous exercise).

2. Create a new Java Application project, call it `AsyncClients`, to avoid confusion with the other, simpler clients. This project, too, is standalone – any artifacts that this web service client will need will be generated by JAX-WS, given the WSDL description for the service.

3. Introduce a web service client reference to the `AuctionManager` web service. When you do this, NetBeans will configure this project to generate all the necessary client-side Java artifacts for this web service.

   Since you created your `AuctionManager` web service in the `EJBWebServices` web application, then the URI to reach that service should be:

   ```
   http://localhost:8080/EJBWebServices/AuctionManagerService
   ```

   Remember to add the suffix `?WSDL` to query the running web service for its WSDL description.

4. This client will need to specify that JAX-WS should generate artifacts that support the new asynchronous API supported by JAX-WS. The simplest way to do this is to use the NetBeans wizard edit to the web service client's attributes, to add this support.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Aula Matriz S.A.S.

Right-click on the node in the `Project` tab that corresponds to the web service client's reference to the web service, under the project node labeled `Web Service References`. Select `Edit Web Service Attributes...`

A dialog will come up with a number of features that can be customized about this client's interaction with the web service – but the specific option to enable the asynchronous API is found under `WSDL Customization`, in the `Global Customization` node: just click on the checkmark next to `Enable Asynchronous Client`.

```
Java EE Development
→ Web Services
→ JAX-WS Web Services
→ Editing Web Service Attributes
```

.

5. Create a new application client for the `AuctionManager` web service, call it `AsyncPlaceBid.java`.

6. Write the client so that it places a bid on an Auction by calling the `placeBid` operation on the `AuctionManager` web service – but have the client perform some work, in between the request that the service place a new bid, and the display of the result of that request. It is enough that the server print out a message, between requesting the placing of the bid and displaying the response.

   Remember that, when you enabled support for asynchronous interactions for this client, you triggered the introduction of two new methods in the JAX-WS web service client proxy for each of the operations in the WSDL description for the web service:

   - `Response<T>` *`operation`*`Async(...)` –
     which returns immediately on invocation, with a `Response<T>` return value that will allow the client to retrieve the result of this web service later – without blocking at the time of the call; and

   - `Future<T>` *`operation`*`Async(..., AsyncHandler<T> h)` –
     which returns immediately on invocation, with an extra parameter for the call which allows the client application to delegate the execution of the work that needs to be done on receiving the response from the web service to the JAX-WS runtime.

   Either of the two could be used to implement the logic requested by the exercise – although the first may be simpler to code...

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Oscar Hernando Olivia Aragon (olivia@hotmail.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Aula Matriz S.A.S.

If you use the NetBeans wizard to insert new code into your lstinline—main()—, you can ask it to help you write a web service call. In this case, it will offer you a list of all the operations available on the `AuctionManager` web service – including the asynchronous variants of each of the operations.

7. Run the client application, to test this asynchronous interaction.

There are a number of ways to capture the requirement the the client be able to use the asynchronous call API:

- Modify the server's WSDL description to include the requirement that all clients support the JAX-WS asynchronous API, if possible.
- Copy the server's WSDL description for this new client to refer to, locally. Modify this client's copy to include the requirement that JAX-WS support asynchronous interactions.
- Write a WSDL customization XML file so that the client's runtime add the requirement that JAX-WS support asynchronous interactions to the WSDL description received from the web service provider.
- Use the IDE to add the requirement that the client support asynchronous interactions to this web service client's artifacts.

Task 2 – Shut Down Web Services

Undeploy the web service provider application(s) you deployed in this exercise, and shut down the application server.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Aula Matriz S.A.S.

Exercise Summary

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

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Aula Matriz S.A.S.

Lab 7

Introduction to RESTful Web Services

Objectives

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

- Understand what RESTful Web Services are.
- Understand the five principles behind RESTful Web Services.
- Understand the advantages and disadvantages of a RESTful approach.

Oscar Hernando Nivia Aragon (olivia@hotmail.com) has a non-transferable license to use this Student Guide.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Aula Matriz S.A.S.

Exercise 1 – Understanding RESTful web services

1. Indicate whether each of the following assertions is true or false:
   (a) RESTful web services are procedural, rpc-oriented services.
   (b) RESTful web services prefer to minimize the number of interactions between client and server, and so prefer to deliver complex data representations.
   (c) RESTful web services commit to XML-based representations for communications.
   (d) RESTful web services prefer stateless interactions.
   (e) RESTful web service requests represent the operation to be performed somewhere in the URI used to invoke that operation on the server side.
2. Choose the statement that best describes the RESTful approach to choosing the right HTTP method to use for any request:
   (a) `GET` is always the preferred method to use.
   (b) `POST` is to be preferred over `PUT`, as it is more flexible.
   (c) `PUT` is to be preferred over `POST`, as it is more precise.
   (d) `POST` and `PUT` have different semantics, so the choice must be based the semantics of the operation.
3. Fill in the HTTP method that corresponds to each of the semantics quoted:

Method	Purpose
______________	Read, possibly cached
______________	Update or create without a known ID
______________	Update or create with a known ID
______________	Remove

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Aula Matriz S.A.S.

Lab Summary

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

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Aula Matriz S.A.S.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Aula Matriz S.A.S.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Lab 8

RESTful Web Services: JAX-RS

Objectives

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

- Implement RESTful web services using JAX-RS
- Deploy REST web services using Jersey, an implementation of JAX-RS

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Aula Matriz S.A.S.

Exercise 1 – Understanding How to Define Web Services Using JAX-RS

1. Indicate whether each of the following statements is true or false:

   (a) Every JAX-RS root resource class must be annotated with a `@Path` annotation.

   (b) If a public method of a resource class does not include an explicit HTTP method annotation, JAX-RS will act as though the code had specified `@GET`.

   (c) Just like web containers do with servlets, the JAX-RS runtime will create an instance of a root resource class the first time the runtime needs that class to handle a client request, then reuse that same instance as other clients require access to the same root resource class.

   (d) A class that inherits from `Application` is mandatory for every server-side application based on JAX-RS, so as to configure the set of resources offered by that application.

   (e) JAX-RS guarantees thread-safety for root resource classes annotated with the `@Singleton` annotation.

2. Two resource methods on the same resource class must differ in one of the following: (Choose all that apply)

   (a) `@Path` URI template.

   (b) HTTP Method.

   (c) Representations produced.

   (d) Representations consumed.

   (e) Visibility.

   (f) Arguments (in number or types).

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Aula Matriz S.A.S.

3. Choose the name of the the type of object that you can inject into a JAX-RS resource class to help it build URIs relative to the current one, to be used as part of responses. (Choose one).

   (a) `PathInfo`

   (b) `UriInfo`

   (c) `Context`

   (d) `Response`

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Aula Matriz S.A.S.

Exercise 2 – Create, Deploy, and Test the ItemManager Web Service

In this exercise, you're going to create, deploy, and test a simple RESTful web service for the Auction application, as a counterpart to the first simple JAX-WS service you implemented before. Once you have your web service ready, you will deploy it to a stand-alone Java VM. Since this is a RESTful web service, you can test it simply using a browser.

Remember that the examples discussed in class are available in your home directory, in the `examples` folder. It may be useful to refer to them, as you create your first RESTful web services.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Task 1 – Define URIs Required by RESTful Service

The simplest service one could imagine using in the Auction project might be an `Item` manager web service, which would allow its clients to create/find/-manage the set of persistent `Item` descriptions used by the system, along the same lines of the JAX-WS `ItemManager` service you wrote before.

1. In a RESTful architecture, the first thing to do is identify the URIs that will be used to refer to the entities to be exposed through the RESTful service. Remember that URIs are to be used to identify the entity of interest, not the operations to be performed with that entity.

2. You do have to choose what operations will be supported by this RESTful service – but in a RESTful architecture, those operations have to be mapped to the standard HTTP methods that one could invoke on of the entities exposed through this service.

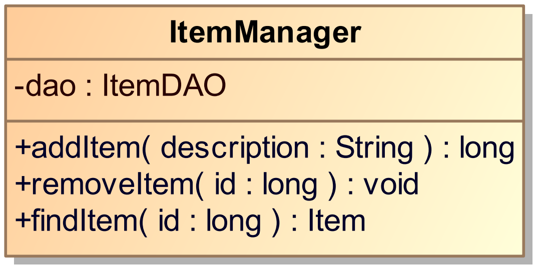

ItemManager
-dao : ItemDAO
+addItem(description : String) : long
+removeItem(id : long) : void
+findItem(id : long) : Item

Figure 8.1: Possible UML Diagram for `ItemManager` Class

As a reminder, Figure 8.1 illustrates the operations that the earlier JAX-WS `ItemManager` web service provided.

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Aula Matriz S.A.S.

Request	HTTP Method	URI
List all Items		
Add New Item		
Update Item		
Get Item		
Remove Item		

Table 8.1: Requests Supported by RESTful Item Manager

Table 8.1 will allow you to capture the RESTful request that will represent the operations that your web service will support.

Consider whether the requests that will accept parameters should describe them as components in the URI path, or query or form parameters to the request.

Task 2 – Define POJO Class

You will implement a JAX-RS `ItemManagerRS` service. It will be responsible for creating new `Item` instances, and finding existing `Item` instances that match a new item. It could also remove instances that are no longer used – but that is a more dangerous service to offer, within the Auction system.

Since you have not yet seen how to pass complex argument into and out of RESTful web services, limit your API to accept primitive types (and Strings), and to return the same: for example, the function to add a new `Item` could return the new `Item`'s id, and the function to remove an `Item` could take the id of the `Item` to remove. You may want to wait to implement the operation to find an `Item` until a later lab.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Aula Matriz S.A.S.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

1. Create a new Java Application project in NetBeans. Call it `RESTfulServices`. Switch to it. You will use this project to create standalone JAX-RS (RESTful) web services.

2. Add the `JAX-RS 1.1` and `Jersey 1.1 (JAX-RS RI)` libraries to the project, to have access to the JAX-RS APIs and Jersey implementation.

   You will also need to add the dependencies that are required to support server-side operations for the Auction application. Since this project will contain web services in charge of manipulating persistent domain instances, the project will have to depend on the right infrastructure: add dependencies on `AuctionApp` (for the domain types), `EclipseLink (JPA 2.0)` for the persistence framework), and `DerbyJDCB` (for the JDBC driver needed to use the Java DB (Derby) database.

3. Create a new class `ItemManagerRS`, in package `labs`. Define the functions that will be required to support the operations you identified above. Implement the business logic associated with each of the functions in the interface for `ItemManagerRS`.

   (a) Create the package `labs` in the `RESTfulServices` project.

   (b) It may be easiest to copy the `ItemManager` class that you wrote originally, as a standalone JAX-WS web service, to this project. Rename the class `ItemManagerRS`, and remove from it any JAX-WS annotations that may be present in the code – although it is possible to have an implementation class that is exposed both as a JAX-WS web service and a JAX-RS service, it will be easier if you concentrate on one kind of web service at a time.

```
Java Development
→ Java Classes
→ Copying Java Classes
```

   You need not copy the `ItemManagerRunner` class from the earlier project.

NetBeans includes a wizard to create JAX-RS (RESTful) web services from scratch – but it is only available in enterprise applications. Since your first project will deploy simple standalone JAX-RS services, you will not be able to call on this NetBeans wizard until a later lab.

Aula Matriz S.A.S.

Task 3 – Incorporate JAX-RS Annotations

Once the `ItemManagerRS` is implement and working properly, it is time to turn it into a RESTful web service provider. Incorporate JAX-RS annotations, to turn your `ItemManagerRS` into a RESTful web service:

1. A `@Path` annotation at class level, to indicate that `ItemManagerRS` is a JAX-RS root resource .

   When/if you click on `Fix Imports`, to have the IDE add the appropriate import for the `@Path` annotation, check that you select the right class name: there is more than one class called `Path` in the Java standard library – the one that represents JAX-RS path templates is `javax.ws.rs.Path`.

2. Add a `@Path` annotation at method level, on each of the methods that is responsible for one of the operations offered by this RESTful web service, to indicate the URI template that triggers calls to this method.

3. Add an HTTP method annotation on each of the methods responsible for one of the operations offered by this service, to indicate which HTTP method is associated with each of these operations.

4. Add JAX-RS parameter annotations, if needed, to pass parameters encoded as part of the request into the web service method call.

   You may need to add `@PathParam` annotations, for instance, to extract parameters that are encoded as part of the URI, and pass them to the function call on the root resource class `ItemManagerRS`.

Task 4 – Deploy Web Service to Standalone JVM

Deploy your `ItemManager` web service in a standalone JVM:

1. Introduce a new class to be your service runner – call it `Runner` – and implement it so as to deploy your web service.

   Remember that, in the simplest case, there is very little to do, to deploy an application that publishes JAX-RS based web services: the JAX-RS runtime will find all the root resource classes that are present, if needed.

   Jersey provides a class called `HttpServerFactory` (which can be found in package `com.sun.jersey.api.container.httpserver` that can set up standalone JAX-RS services in a JVM.

2. Ensure that the database server is running.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Aula Matriz S.A.S.

3. Execute the `Runner` application just created.

```
Java Development
→ Java Classes
→ Executing Java Programs
```

Task 5 – Test Web Service Using Browser

RESTful web services represent web service interactions in terms of simple HTTP requests and responses. For simple kinds of interactions (those that are HTTP `GET` requests, or simple `POST` requests), a web browser is enough to generate the requests that the RESTful web service will process.

Use a browser window to generate `GET` requests to test your RESTful web service, according to the URI templates and HTTP methods you enumerated in Exercise 2.

Generating a `POST` request from a browser requires an HTML page with a form in it that the browser can submit – in the form, you could specify that the method to submit the form will be `POST`.

Code 8.1 shows an example of a simple HTML page that will use `POST` to submit a hardcoded URI. Code 8.2 shows another example of an HTML page that will allow the user to enter information to include as part of the request submitted.

```
<html>
  <body>
    <form action="/jaxrs/items/Guitar" method="POST">
      <input type="submit"/>
    </form>
  </body>
</html>
```

Code 8.1: Simple HTML Form with Hardcoded Request

You will find a couple of examples of HTML files that you could use to test your RESTful web services in the `resources` folder in your home directory.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Aula Matriz S.A.S.

```
<html>
  <body>
    <form action="/jaxrs/items" method="POST">
      <input type="text" name="description"/>
      <input type="submit"/>
    </form>
  </body>
</html>
```

Code 8.2: HTML Form with Interactive Data

Task 6 – Shut Down Web Service

Shut down the `Runner` application.

```
Java Development
→ Terminating a Running Process
```

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Aula Matriz S.A.S.

Exercise 3 – Create, Deploy, and Test the UserManager Web Service

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

In an earlier lab, you built a `UserManager` web service using the "contract-first" approach. A RESTful architecture does not have an equivalent notion – there is no standard description of what a web service will provide, written independently of and before implementing the web service.

In this exercise, you will implement the RESTful counterpart to the `UserManager` service you implemented before. The activities that the web service would need to support, as far as user management is concerned, include:

- Add new users to the application. This might involve specifying the user's login id, full name, and email address. It could also include a password, to authenticate the user.
- Update information about an existing user, such as their full name, or their email address.
- List all users, or just the users that match some criteria.
- Remove users from the application.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

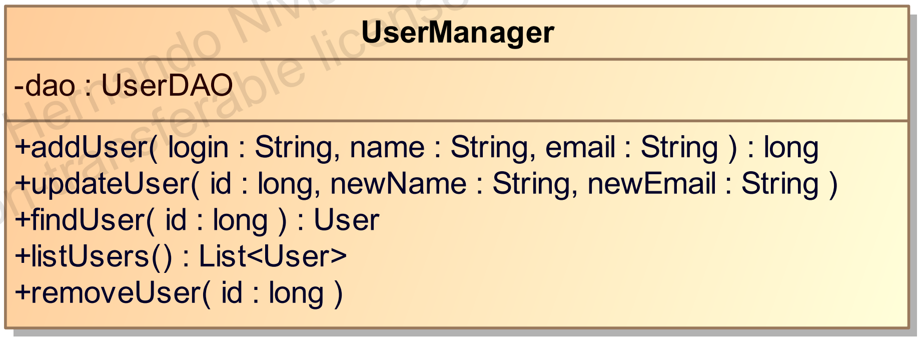

UserManager
-dao : UserDAO
+addUser(login : String, name : String, email : String) : long
+updateUser(id : long, newName : String, newEmail : String)
+findUser(id : long) : User
+listUsers() : List<User>
+removeUser(id : long)

Figure 8.2: Possible UML Diagram for `UserManager` Class

Figure 8.2 illustrates what the UML for a class that provides such operations could look like.

Since you have not yet learned how to pass complex types to or from a RESTful web service, you can delay support for listing all users until a later lab.

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Aula Matriz S.A.S.

Request	HTTP Method	URI
List all Users		
Add New User		
Update User		
Get User		
Remove User		

Table 8.2: Requests Supported by RESTful User Manager

Task 1 – Define URIs Required by RESTful Service

The `UserManagerRS` web service will allow its clients to create/find/manage the set of persistent `User` instances representing users known by the system:

1. In a RESTful architecture, the first thing to do is identify the URIs that will be used to refer to the entities to be exposed through the RESTful service. Remember that URIs are to be used to identify the entity of interest, not the operations to be performed with that entity.

2. You do have to choose what operations will be supported by this RESTful service – but in a RESTful architecture, those operations have to be mapped to the standard HTTP methods that one could invoke on of the entities exposed through this service.

Table 8.2 will allow you to capture the RESTful request that will represent the operations that your web service will support.

Consider whether the requests that will accept parameters should describe them as components in the URI path, or query or form parameters to the request.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Aula Matriz S.A.S.

Task 2 – Define POJO Class

1. Create a new class `labs.UserManagerRS`, in the `RESTfulServices` project. Define the functions that will be required to support the operations you identified above.

2. Implement the business logic associated with each of the functions in the interface for `UserManagerRS`.

Although you implemented your JAX-WS `UserManager` web service using the "contract-first" approach, you were still responsible for capturing the business logic associated with all web service operation: you wrote a `UserManagerImpl` class, to do all the business-level work.

The simplest way to start writing the RESTful version of this service in class `UserManagerRS` may be to copy that earlier `UserManagerImpl` class into this project, rename it, and remove all JAX-WS related annotations from the new `UserManagerRS` copy of that class.

Task 3 – Incorporate JAX-RS Annotations

Once the `UserManagerRS` is implement and working properly, it is time to turn it into a RESTful web service provider. Incorporate JAX-RS annotations, to turn your `UserManagerRS` into a RESTful web service:

1. Add a `@Path` annotation at class level, to indicate that `UserManagerRS` is a JAX-RS root resource .

2. Add a `@Path` annotation at method level, on each of the methods that is responsible for one of the operations offered by this RESTful web service, to indicate the URI template that triggers calls to this method.

3. Add an HTTP method annotation on each of the methods responsible for an operation offered by this service, to indicate which HTTP method is associated with each of these operations.

4. Add JAX-RS parameter annotations, if needed, to pass parameters encoded as part of the request into the web service method call.

Task 4 – Deploy Web Service to Standalone JVM

Deploy your `UserManagerRS` web service to a standalone JVM. There are at least two ways to do this:

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Aula Matriz S.A.S.

- Add a `main()` function to your class, and use it to deploy your web service.
- Introduce a new class to be service deployer – call it `Runner` – and implement it so as to deploy your web service.

Note that, if you implemented a class `Runner` separate from your `ItemManagerRS` in order to deploy that service in the earlier exercise, there may be nothing else to do now: the `Runner` class might have just relied on the JAX-RS runtime to find and deploy the root resources available to the application; if you simply run that application again now, it will find both root resources in this project, and it will deploy both.

Task 5 – Test Web Service Using Browser

RESTful web services represent web service interactions in terms of simple HTTP requests and responses. For simple kinds of interactions (those that are HTTP `GET` requests, or simple `POST` requests), a web browser is enough to generate the requests that the RESTful web service will process.

Use a browser window to generate `GET` or `POST` requests to test your RESTful web service, according to the URI templates and HTTP methods you enumerated in Exercise 2.

Task 6 – Shut Down Web Service

Shut down the `Runner` application.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Aula Matriz S.A.S.

Lab Summary

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

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Aula Matriz S.A.S.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Lab 9

JAX-RS Web Service Clients

Objectives

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

- Understand how to create web service clients using JAX-RS.

Oscar Hernando Nivia Aragon (olivia@hotmail.com) has a non-transferable license to use this Student Guide.

Aula Matriz S.A.S.

Exercise 1 – Understanding RESTful clients

1. Indicate whether each of the following statements is true or false:

   (a) Clients of a RESTful service that are implemented using the class `java.net.URL` in Java are limited to HTTP `GET` requests, because the `URL` class has that limitation.

   (b) `java.net.URL` cannot pass an entity body as part of any request.

   (c) The machinery associated with `java.net.URL` can automatically encode and decode query parameters and returns encoded in `Base-64`.

   (d) The machinery provided by the Jersey Client API can automatically encode and decode query parameters and returns encoded in `Base-64`.

2. Choose the type used to represent RESTful resources in the Jersey Client API: (Choose one).

   (a) `Resource`

   (b) `Client`

   (c) `WebResource`

   (d) `URL`

3. To obtain metadata about the most recent interaction with the web service, the Jersey Client API requires that:

   (a) The client specify that the expected type for the incoming payload be `ClientResponse`.

   (b) The client call a `getMetadata()` API in `WebResource`.

   (c) The client call the `WebResource` HTTP method within a `try/catch`, because the metadata will be part of the exception that the Jersey runtime will throw to the application.

   (d) The client gain access to the underlying `HTTPUrlConnection` instance because HTTP metadata is not exposed through the Jersey Client API.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Aula Matriz S.A.S.

Exercise 2 – Building Web Service Clients

You have implemented two different RESTful web services: `ItemManagerRS` and `UserManagerRS`. In this lab, you will build web service clients for those web services as stand-alone applications, using the Jersey client APIs.

Task 1 – Create New Project

To avoid confusion, create a new Java Application project (call it `RSClients`), to place your new standalone client applications into. This new project should depend on the `AuctionApp` project, since JAX-RS does not generate any artifacts for you: if a web service were to require, or return, an application type, you would need to provide that type. The project will also need to depend in the `JAX-RS 1.1` and `Jersey 1.1 (JAX-RS RI)` libraries.

At the moment, this may not be important, since your web services may be written in terms of parameters and return types that are primitives or instances of `String`. In general, however, you would be communicating with these web services using application types.

Task 2 – Build a Web Service Client for ItemManagerRS

The `ItemManagerRS` web service offers several operations, like: add new items to the persistent store, and remove existing items. A single web service client that is a standalone application may require then several parameters: the first one could be the name of the operation to perform (one of `add`, or `remove`), the others any information that may be required by each of those operations.

1. Write a standalone Java application (call it `clients.ItemManagerClient`) that accepts parameters, and uses them to invoke the appropriate operation on the `ItemManagerRS` web service.

   To call the RESTful web service, use the Jersey client API. This involves:

   - Obtaining an instance of `Client`.
   - Obtaining instances of `WebResource` that represent the resources to contact.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.
Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Aula Matriz S.A.S.

- Invoke operations on those resources through their `WebResource`.

  To invoke operations on a `WebResource`, you would call the appropriate HTTP method on that instance: `WebResource` provides `get()`, `put()`, `post()`, and so on, to issue requests to the web service using the corresponding HTTP method.

  Before you invoke that HTTP method operation, though, you may need to call other methods provided by the `WebResource` API, to specify (among others):

  - Any query parameters to include in the request.
  - Any preferred return encoding.
  - The outgoing encoding to use for the body of the request, if any.

As an alternative, you could write several standalone web service client applications, one for each operation supported by the `ItemManagerRS` web service: each client application would then be hardcoded to invoke a specific operation.

2. Run the client application(s), to test all the operations in `ItemManagerRS`.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Aula Matriz S.A.S.

Task 3 – Build a Web Service Client for UserManager

The `UserManagerRS` web service offers several operations, like: add new users to the persistent store, and remove existing users.

1. Write a standalone Java application (call it `UserManagerClient`) that accepts parameters, and uses them to invoke the appropriate operation on the `UserManagerRS` web service – along the lines of the client application(s) you wrote earlier, to interact with the `ItemManagerRS` web service.

2. Run the client application(s), to test all the operations in `UserManagerRS`.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Aula Matriz S.A.S.

Exercise Summary

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

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Aula Matriz S.A.S.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Lab 10

JAX-RS and JavaEE

Objectives

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

- Deploy RESTful web services to a web container.
- Secure RESTful web services in a web container, delegating authentication to the container.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Aula Matriz S.A.S.

Exercise 1 – Understanding RESTful resources in a container

1. Indicate whether each of the following statements is true or false:

   (a) Deploying RESTful JAX-RS resources in a web application requires changes to the `web.xml` configuration file.

   (b) Deploying RESTful JAX-RS resources in a web application requires the presence of a subclass of `Application`.

   (c) Security constraints that apply to a JAX-RS resource can only be captured in the `web.xml` configuration for the web application that the resource is deployed in.

   (d) Declarative annotation of security constraints as annotations on the resource class automatically configures the web container's login configuration for this application.

   (e) Declarative authorization constraints can be specified in terms of specific users of the application.

   (f) The JAX-RS and JAX-WS specifications are not compatible, so the only way to have web services that can be reached via either REST or SOAP is to have two separate implementation classes, one implemented in JAX-RS, and the other in JAX-WS.

   (g) A JAX-RS resource that is implemented as a Singleton EJB must be made thread-safe programmatically at application level.

2. Choose the types that can be injected into a JAX-RS resource class to obtain information about the caller of a service: (Choose as many as appropriate).

   (a) `SecurityContext`

   (b) `ServletContext`

   (c) `UriInfo`

   (d) `ServletRequest`

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Aula Matriz S.A.S.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Exercise 2 – Deploy RESTful Web Services to a Web Container

In Lab 8, you built two simple web services, `ItemManagerRS` and `UserManagerRS`. They were both designed to be simple RESTful POJO web services, and they were both deployed standalone in a JVM. In this exercise you will deploy them into a web application, so as to leverage the functionality provided by the web container: you will be able to authenticate clients of the web service by delegating the authentication logic to the web container.

Task 1 – Copy Existing Web Services to Web Application

RESTful POJO web services can be deployed either standalone, using the HTTP server built into Java SE 6, or to any web container. The implementation of the POJO web service itself does not have to change, due to the environment it will be deployed into – any differences at that level are handled by the JAX-RS (Jersey) runtime.

1. To avoid confusion, create a new Web Application project (give it the name `RSInContainer`) to deploy the two web services into. This project will need to depend on the `AuctionApp` project, and all the usual suspects, for a server-side application.

   `RSInContainer` needs to depend on `AuctionApp`, since the project will include web service implementation classes that rely on the back-end logic (domain types and persistence layer) provided by `AuctionApp`. Since that persistence layer is implemented using JPA, the `RSInContainer` project needs to depend on both the `EclipseLink (JPA 2.0)` and Derby JDBC libraries.

   In principle, `RSInContainer` also needs to depend on the `JAX-RS 1.1` and `Jersey 1.1 (JAX-RS RI)` libraries, since RESTful services are part of the project. It won't hurt to specify them, too. However, you will be deploying to GlassFish as your web container, and GlassFish has these already built-in.

2. Add to this project two server resource configuration entries:

   (a) Configure a database connection pool, so that this application (and, in particular, the persistence layer that your services rely on) can access the database efficiently. You can name this pool `auctionDbPool`, and use the NetBeans connection to the Auction database that you configured in in the setup lab, so that NetBeans can configure the connection pool.

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Aula Matriz S.A.S.

(b) Configure a JDBC resource, so that components within this project can access the database through the `auctionDbPool` via JNDI lookup. You should name this JDBC resource `jdbc/auction`.

You can refer to the instructions for Lab 4 Exercise 2 Task 1: you had to set up a similar configuration for another web application then.

3. Copy the code for the two services `ItemManagerRS` and `UserManagerRS` from the `RESTfulServices` project you created in Lab 8 into this new project.

   The simplest way to copy the two services is:

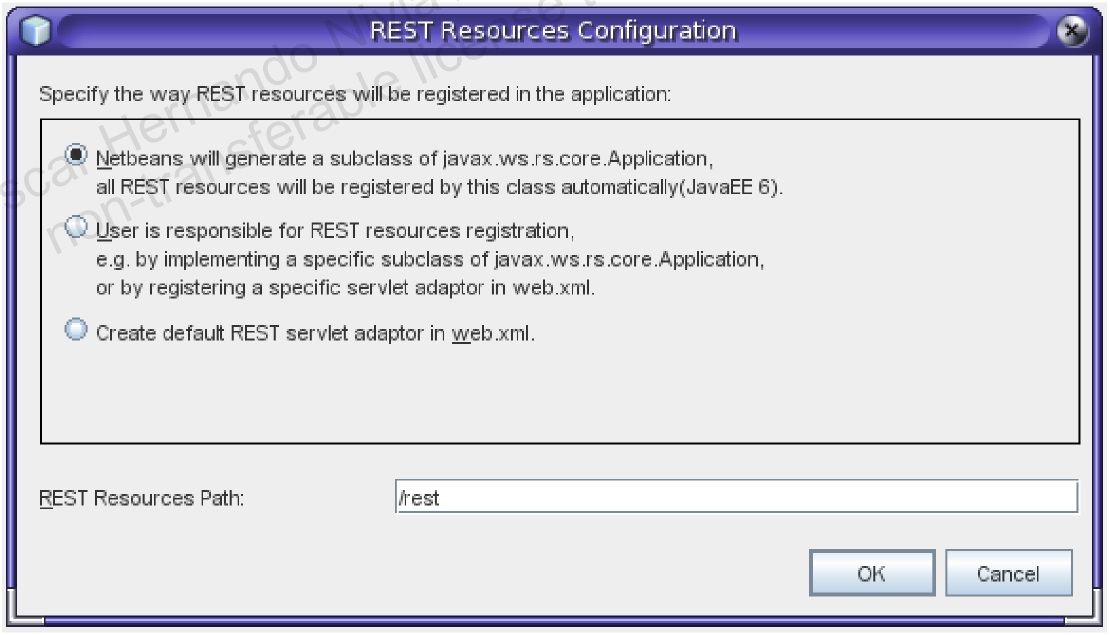
   (a) Select the `labs` package from the original `RESTfulServices` project.

   (b) Right-click on that `labs` node, and select `Copy`.

   (c) Select the `Source Packages` node within the `RSInContainer` project.

   (d) Right-click on that `Source Packages` node, and select `Paste`.

When you copy the two JAX-RS implementation classes into the `RSInContainer` project, NetBeans will note that you are importing JAX-RS resources: a dialog will pop up, asking you how NetBeans should proceed with the configuration of those resources within the web application. Figure 10.1 illustrates the dialog that you will see.



REST Resources Configuration

Specify the way REST resources will be registered in the application:

(●) Netbeans will generate a subclass of javax.ws.rs.core.Application, all REST resources will be registered by this class automatically(JavaEE 6).

( ) User is responsible for REST resources registration, e.g. by implementing a specific subclass of javax.ws.rs.core.Application, or by registering a specific servlet adaptor in web.xml.

( ) Create default REST servlet adaptor in web.xml.

REST Resources Path: /rest

Figure 10.1: Automatic JAX-RS Configuration Dialog on Import

You should go ahead and let NetBeans configure these resources automatically – just change the resource path that it will configure the resources with to be

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.
Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Aula Matriz S.A.S.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

/rest, to match the instructions in the next step.

A new node named RESTful Web Services will appear in the RSInContainer project: it contains descriptions of the RESTful services that NetBeans knows to exist within the project.

Task 2 – Configuring Web Services Within Web Application

No explicit configuration changes to the web application (that is, changes to the web.xml file) are needed – JAX-RS will configure URLs to access the web services within the application by default, based on their @Path annotations. The URLs will have, by default, the form:

```
http://localhost:8080/RSInContainer/pathInAnnot
```

However, once RESTful web services are part of a web application, it may be necessary to distinguish incoming URL requests that are supposed to be handled by JAX-RS services from URLs that are to be processed by the web application front end. An Application class can be used to capture explicitly the set of RESTful resources to be offered by the application; it can also be used to specify a prefix to URLs that are to be processed by those resources, to distinguish RESTful requests from web front end requests, through a @ApplicationPath annotation.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide

If you selected to let the user register JAX-RS resources when you copied the two resource classes into the application, you will need to :

1. Create a class that inherits from Application – you could call your subclass labs.AppResources. Implement it so as to specify the set of RESTful resources provided by this web application: the two root resources ItemManagerRS and UserManagerRS.

   The Jersey runtime will look for all resources available within the web application by default, even with an Application class also present – which means that you could leave your subclass as an empty function.

   To retain more precise control over which resources this application will export, you would need to add the method:

   ```
   public Set<Class<?>> getClasses()
   ```

   to your Application subclass, and make it return the set of resources to export: in this case, the two classes ItemManagerRS and UserManagerRS.

2. Annotate your child class with an @ApplicationPath annotation, to set the prefix that distinguishes RESTful requests to /rest.

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Aula Matriz S.A.S.

If you let NetBeans register your resources automatically, then the class it created will be found within a node called `Generated Sources (rest)` within the `RSInContainer` project. It will also include an `@ApplicationPath` annotation, to specify the prefix that will distinguish URIs that refer to RESTful services within this application.

As a result of this `@ApplicationPath` annotation, URL requests for the two web services will have the form:

```
http://localhost:8080/RSInContainer/rest/pathInAnnot
```

Task 3 – Deploy and Test Services in Web Application

Deploying the `RSInContainer` web application to a web container will automatically deploy the RESTful web services contained within it.

To test the services deployed within this web application, you could do the same things you did to test the original standalone services in Labs 8 and 9:

- Use a web browser to issue `GET` and `POST` requests.
- Write simple Java client applications that use the Jersey Client API to invoke operations on these services.

In either case, make sure that the resource URIs that you use to invoke the services are a match for the URIs that the services within the web container expect. To be sure, shut down (if you have not yet done so) the standalone JAX-RS service applications that you used in Lab 8 to deploy the original version of these services.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Aula Matriz S.A.S.

Exercise 3 – Secure POJO Web Services in Web Container

Earlier, you deployed the two web services `ItemManagerRS` and `UserManagerRS` to a web container, as part of a web application. One of the advantages of doing this is that it becomes possible to secure those web services, delegating authentication to the web container. In this exercise, you will secure access to the `UserManagerRS`, so that only administrators can create new users.

Task 1 – Define Security Constraints on Web Service

You have to capture, as part of the description of the service, that there are two kinds of users of the `UserManagerRS` web service, "administrators" and everyone else, and that only administrators are allowed to add new users to the application.

The way to configure security constraints in JAX-RS web services borrows the same annotations that JAX-WS (and before it, EJBs) use to capture these constraints.

1. For convenience, let's define two *roles*, or groups of users, called `user` and `administrator`. The set of roles expected by a JAX-RS web service can be captured by annotating the web service implementation class with a `@DeclareRoles` annotation, listing those roles by name.

2. A `@RolesAllowed` annotation can be specified at class level, to specify a default requirements that the caller belong to certain roles to call any operation on the web service implemented by that class. It can also be specified at method level, to restrict access to a particular operation in that class. Capture the constraints that callers in the `user` role can call any of the operations in either `ItemManagerRS` or `UserManagerRS`, but only callers in the `administrator` role can add users through the `UserManagerRS` web service.

You could also log the name of the person adding a new user to the container's log file, just to show that authenticated user information is provided to the web service.

Task 2 – Define Security Constraints on Web Application

Capturing authentication constraints at the level of the web service implementation classes isn't enough. JAX-RS will delegate authentication to the web con-

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Aula Matriz S.A.S.

tainer – but the web container requires additional configuration information, in order to perform authentication.

The configuration of the web application's `web.xml` to support authentication for JAX-RS web services is also a match for the configuration required for JAX-WS web services – with one exception: JAX-RS web services can be triggered by requests using any of the available HTTP methods.

1. A `login-config` element has to be added to the web application's configuration file, `web.xml`. This element is used to configure the authentication mechanism that the web container will use. Set it to `BASIC`, which expects clients to provide userid and password information in an HTTP header.

2. A `security-constraint` element has to be added to `web.xml`, too. This element indicates which URLs are to require user authentication, for a given HTTP method, and which roles a caller has to belong to, in order to be allowed access to the resource. When describing which URLs are protected, look up which HTTP methods are expected by your RESTful services. The URIs to be protected all begin with the prefix `/rest`, due to the `@ApplicationPath` annotation you added above.

3. The set of users expected by the application, along with the roles that each user will belong to, must be specified, too - but this is not part of the product-independent web application configuration.

   In the case of GlassFish in particular, the product-specific configuration file is `sun-web.xml`. Add `security-role-mapping` elements to this file, for each of the users that will require authentication to use the application.

As a starting point, you could copy the matching configuration elements from the configuration files of the `POJOsInContainer` web application, and paste them into the corresponding locations in the `RSInContainer` configuration files. Some of the elements may require editing – in particular, the URIs and HTTP methods that are to be protected – but it would reduce the amount of work involved.

Task 3 – Configure Users in Authentication Realm

The actual authentication of principals is outside the scope of the specification of a web application – it is a responsibility of the web container. Therefore, configuration of the information associated with known principals (for example, their user names and password) must be done at the level of the web container.

Since you already configured a number of users in an earlier lab, let's just continue to use the same set of users as before.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Aula Matriz S.A.S.

Task 4 – Building Authenticating Web Service Clients

In an earlier lab, you created two standalone applications that use the Jersey client API to interact with RESTful services, which you may have called `ItemManagerClient` and `UserManagerClient`. You could use those two applications to test the web services that can be deployed as part of the `RSInContainer` web application – if they would just authenticate themselves to the web service:

1. Copy the two client applications into two new applications, which you could call `AuthItemManagerClient` and `AuthUserManagerClient`.

2. Modify both client applications so that the URI used to contact the web service matches the URIs associated with the web services deployed to the `RSInContainer` web application.

   Remember that the URIs for the services deployed with the `RSInContainer` will look like this:

   ```
   http://localhost:8080/RSInContainer/rest/pathInAnnot
   ```

3. Modify the two new applications so that they provide authentication information, when they issue requests to their web services.

   Remember that the way to embed authentication information into web service requests issued through the Jersey Client API is to associate an *authentication filter* with the `Client` instance used by the client applications. This authentication filter will be instantiated with a fixed userid and password, which will then be attached to all outgoing requests associated with that `Client` instance.

Task 5 – Deploy and Test Services in Web Application

1. Deploy the `RSInContainer` web application. This will deploy the web application, including all RESTful web services configured by the `Application` subclass in that web application.

2. Run your two authenticating client applications, to test that the web services are working correctly, and limiting access to the right users.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Aula Matriz S.A.S.

Lab Summary

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

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Aula Matriz S.A.S.

Lab 11

Implementing More Complex Services Using JAX-RS

Objectives

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

- Pass complex objects to and from a web service.
- Map Java exceptions from a web service provider to SOAP faults.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Aula Matriz S.A.S.

Exercise 1 – Understanding Features of JAX-RS

1. Indicate whether each of the following statements is true or false:

   (a) Jersey only supports XML representations out of the box, through Java's built-in JAXB technology. Other representations have to be added as application-level extensions, via classes that implement the two JAX-RS interfaces `MessageBodyReader` and `MessageBodyWriter`.

   (b) Any one resource method can only support one representation type.

   (c) `Response` objects are used to capture metadata about the return value to be delivered to the caller.

   (d) A resource method that needs to return metadata to a caller, when an exception is thrown within the resource method, must be defined to return a `Response` instance – so the method can return the appropriate metadata.

   (e) When creating a custom representation for an application-level type, both `MessageBodyReader` and `MessageBodyWriter` implementations for that application-level type must be provided.

   (f) Instances of entity provider types – classes that implement the two interfaces `MessageBodyReader` and `MessageBodyWriter` – are created and thrown away after each use.

2. Choose the class-level annotation that must be used for any classes that implement `MessageBodyReader`, `MessageBodyReader`, or `ExceptionMapper`: (Choose one).

   (a) `@Context`

   (b) `@Path`

   (c) `@Provider`

   (d) `@Resource`

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Aula Matriz S.A.S.

3. How can you identify sub-resource locator methods? (Choose the one best match).

   (a) They are methods in a resource class that have no `@Path` annotation.

   (b) They are methods in a resource class that have no HTTP method annotation.

   (c) They are methods in a resource class that are annotated with `@Resource`.

   (d) They are private methods in a resource class.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Aula Matriz S.A.S.

Exercise 2 – Pass Complex Types

The two RESTful web service you have implemented so far, `ItemManagerRS` and `UserManagerRS`, have been limited by the fact that you had only seen how to pass simple types. In this Module, you learned how to pass complex types, and so you can now make these two services more complete.

You may work on either the `RESTfulServices` project (for standalone POJO web services deployed to a JVM), or the `RSInContainer` project (for POJO web services deployed to a web container).

Task 1 – Complete the API of the Existing Web Services

Complete the APIs of your two web services, to include those operations that require complex types as output parameters:

1. Add support for a `findItem` operation to your `ItemManagerRS`, which accepts an id as a parameter, and returns the `Item` with that id.
2. Add support for a `findUser` operation to your `UserManager`, which accepts an id as a parameter, and returns the `User` with that id.

When you first defined the two services, you considered what URL templates and HTTP methods would be appropriate for each of the operations that the RESTful service would support on the resources that would be offered. The methods that you add here will need to be annotated with `@Path` and parameter annotations to match.

Task 2 – Update Client Applications

You will need to update your client applications to include support to invoke the new operations just added to the `ItemManagerRS` and `UserManagerRS` web services.

Task 3 – Deploy and Test Services in Web Application

Deploy the updated web services, and test them using your updated client applications.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Aula Matriz S.A.S.

Exercise 3 – Improve Web Service Responses

RESTful web services are expected to follow closely the model initially defined by the HTTP standards for web interaction. In particular, resource responses are expected to carry status codes along with any payload that may be intended for the client, in order to give the client more information (*meta-data*) about the request just processed.

In this exercise, you will update your JAX-RS `UserManagerRS` web service to provide that additional metadata:

- Requests to add users ought to return a `201` status code ("Created"), along with the URI where the client will find the entity just created, in the future. In addition, the server could return the actual entity created back to the client, as the payload of the return message – instead of just the id associated with the new entity, as you had done before.
- Requests to update users ought to return a `404` status code ("Not Found") if the user that is supposed to be updated cannot be found.

  Such requests could also return a `304` status code ("Not Modified") if the update operation did not actually change the user specified.

You may modify your `UserManagerRS` web service in either the `RSInContainer` or `RESTfulServices` projects. The version of `UserManagerRS` deployed as part of the `RSInContainer` web application already has some of this more sophisticated metadata – the authentication support provided through the web container will result in responses with a `401` ("Not Authorized") status code – so it may be a better choice for enhancing even further.

1. Modify the implementation of the method in `UserManagerRS` that handles requests to add new users so that it returns the `201` status code, along with the location of the new resource.

   To control metadata associated with the response from a JAX-RS method, the method must return an instance of `Response`. By manipulating the instance of `Response` to be returned, the application can control the response metadata.

   `Response` offers methods to specify some standard response types (`ok()`, `created()`, `notModified()`, among others) , along with the more generic `status()`. When some of those status codes have associated metadata, the corresponding method accepts as parameters the information that is expected: `created()`, for instance, expected the URI where the entity just created is found as a parameter, in order to encode that URI as a response header for the client to read.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Aula Matriz S.A.S.

The simplest way to create the URI that is expected as the parameter to `Response.created()`, you could take the easy out and simply hardcode the URI template to instantiate. A better solution would involve using `UriInfo` and `UriBuilder` to figure out what the URI should be dynamically. `UriInfo` can be obtained via injection, and `UriBuilder` is obtained from the `UriInfo` instance.

2. Modify the implementation of the method in `UserManagerRS` that handles requests to update users so that it returns "Ok" or "No Found", as appropriate. You may also have it return "Not Modified", when applicable.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Aula Matriz S.A.S.

Lab Summary

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

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Aula Matriz S.A.S.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Aula Matriz S.A.S.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Lab 12

Trade-Offs Between JAX-WS and JAX-RS Web Services

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

- Discuss the trade-offs involved in the choice to implement a web service using either JAX-WS or JAX-RS technology.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Aula Matriz S.A.S.

Exercise 1 – Understanding the difference between JAX-WS and JAX-RS technologies

1. Choose the qualifier that best describes each of the two types of web services in the table below. (Choose one row for each column).

	SOAP services	**REST services**
Activity-oriented		
Method-oriented		
Resource-oriented		
URI-oriented		

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Aula Matriz S.A.S.

Exercise 2 – Building More Complex Web Services Using JAX-RS

The web services you are implementing are there to support an Auction application. In this exercise, you will write some of the higher-level services associated with such an application, as RESTful services.

For reference purposes, Figure 12.1 presents a possible UML diagram for the JAX-WS implementation of this service that you built earlier in the course. The RESTful implementation of the web service may organize its API differently – but the functionality ought to be equivalent to that of the earlier JAX-WS version of the service.

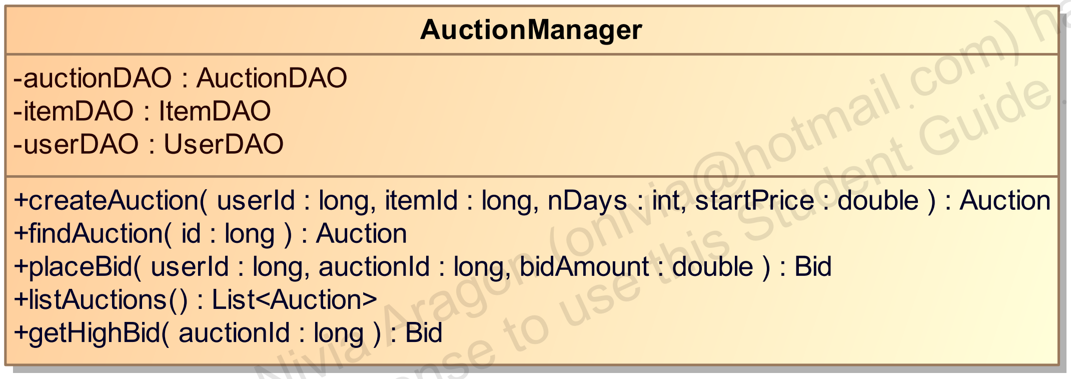

AuctionManager
-auctionDAO : AuctionDAO
-itemDAO : ItemDAO
-userDAO : UserDAO
+createAuction(userId : long, itemId : long, nDays : int, startPrice : double) : Auction
+findAuction(id : long) : Auction
+placeBid(userId : long, auctionId : long, bidAmount : double) : Bid
+listAuctions() : List<Auction>
+getHighBid(auctionId : long) : Bid

Figure 12.1: Possible UML Diagram for AuctionManager Web Service

Task 1 – Define URIs Required by RESTful Service

The goal for the Auction application was to support the ability to manage auctions – so you need to provide an Auction service, which you will now implement as a RESTful service. An Auction application must support, at a minimum, the ability for users to create and look up auctions, and place bids on existing auctions.

1. In a RESTful architecture, the first thing to do is identify the URIs that will be used to refer to the entities to be exposed through the RESTful service. Remember that URIs are to be used to identify the entity of interest, not the operations to be performed with that entity.

2. You do have to choose what operations will be supported by this RESTful service – but in a RESTful architecture, those operations have to be mapped

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.
Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Aula Matriz S.A.S.

to the standard HTTP methods that one could invoke on of the entities exposed through this service.

Consider whether the requests that will accept parameter should describe them as components in the URI path, or query or form parameters to the request.

Task 2 – Implement a AuctionManager Web Service

1. Define a `AuctionManager` Web Service; you could call it `AuctionManagerRS`.
2. Include support for the operations you identified above.

   The `AuctionManagerRS` web service must be implemented as a JAX-RS root resource, with the proper `@Path` and HTTP method annotations on each of the methods that will support those RESTful operations.

Task 3 – Implement a RESTful Client Application

You may be able to test some of the methods on this new Auction service simply by using a browser. In order to test all the services, however, it may be best to implement a client application that uses the Jersey client API to invoke all the operations provided by the `AuctionManagerRS` web service.

Task 4 – Deploy and Test Services in Web Application

Deploy the new Auction web service, and test it using your auction client application.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Aula Matriz S.A.S.

Lab 13

Web Services Design Patterns

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

- Decide when to apply web services-based design patterns .

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Aula Matriz S.A.S.

Exercise 1 – Understanding Web Services Design Patterns

1. Which pattern has the following design goals:
   - Decouple the input and the output messages.
   - Deliver the output message from the server to the client.
   - Associate an output message to the corresponding input message.

   Choose one:

   (a) Asynchronous Interaction

   (b) JMS Bridge

   (c) Web Service Cache

   (d) Web Service Broker

2. Consider a simple design of an application that wants to integrate two remote services into its own workflow. Such a simple design assumes that a transaction begun by the client automatically propagates to both remote services. This allows both remote services to join that same transaction. When they have done so, a failure during processing in the second remote service allows it to roll back its own state by rolling back the current transaction. However, this automatically rolls back the work done by the first remote service, and any work done by the client, during that same transaction.

   Which pattern would you implement in the above scenario to support the transactional behavior of the application? (Choose one).

   (a) JMS Bridge

   (b) Web Service Cache

   (c) Web Service Broker

   (d) Web Service Logger

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Copyright 2010 Sun Microsystems, Inc. All rights reserved. DWS-4050-EE6 Rev. A.0

Aula Matriz S.A.S.

Lab 14

Best Practices and Design Patterns for JAX-WS

There is no exercise for this module.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Aula Matriz S.A.S.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Aula Matriz S.A.S.

Lab 15

Best Practices and Design Patterns for JAX-RS

There is no exercise for this module.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Oscar Hernando Nivia Aragon (onivia@hotmail.com) has a non-transferable license to use this Student Guide.