Lab 11: EJBs and Web Services

In this lab, you will expose an EJB as an interoperable Web service. You will then write a simple client.

Objectives:

- To work with EJB-based Web Services

Part 1: Creating the Service

Steps:

_1. First, you need to create an EJB project for your EJB. Follow these steps:

   a. Choose **File - Other - EJB - EJB Project** to start the wizard.

   b. On the first wizard page, for the *Project name*, enter **lab11EJB**.

      At the bottom of the wizard page, put a checkmark next to the *Add project to an EAR* box, then for the *EAR Project Name*, enter **lab11EAR**.

      Press Next.

   c. On the next wizard page, note the defaults, then press Next.

   d. On the next wizard page, uncheck the box labeled *Create an EJB Client JAR Module...*, then press Finish.

      You can decline any invitation to change perspectives, and close any Technology Quickstart window.

   e. You should see the new EJB and EAR projects in the Enterprise Explorer view. Note that it's OK to have an error at this point since you haven't yet created any EJBs in the project.

_2. In the Enterprise Explorer view, expand the *lab11EJB/lab11EJB* entry -- you should see an empty folder for *Session Beans*.

_3. Right-click on the *Session Beans* folder and choose **New - Session Bean (EJB 3.x)**. Complete the wizard:

   a. On the first wizard page, for the *Java package*, enter **ship**.

      For the *Class name*, enter **ShippingBean**.

      Note that RAD creates the *local* business interface by default, but for this lab, we will use the Remote interface. Uncheck the *Local* box and put a checkmark next to the *Remote* box, then, for the remote interface name, enter **ship.ShippingRemote** and press Next.

   b. On the next wizard page, uncheck the boxes for *Inherited abstract methods* and *Constructors from superclass*, then press Finish.

   c. RAD creates the business interface and the implementation class and opens the class into the Java editor.

   d. Look in the Enterprise Explorer and note that you can see the source files under the *ejbModule* folder in the *ship* package.

_4. Double-click *ShippingRemote.java* to open it into the Java editor. Note how the wizard inserted the @Remote annotation to configure this as a remote interface.

   In the interface, define the *calcDomestic* method that returns a hypothetical shipping-charge calculation:

Copyright © Descriptor Systems 2014. All Rights Reserved. Materials may not be reproduced in any form without prior written consent of Joel Barnum.

```
public double calcDomestic(String zip);
```

Press Ctrl+S to save and compile. You should have no errors in the interface, but you will have errors in the implementation class.

_5. Open ShippingBean.java (the class). Note how the wizard inserted the @Stateless annotation to configure the EJB as a stateless session bean. In the class, write a simple implementation method that returns a hard-coded shipping charge:

```
public double calcDomestic(String zip)
{
    return 3.55;
}
```

Press Ctrl+S to save and compile.

_6. Now let's do a quick test to see if the EJB functions properly, though it's not yet a Web service:

a. In the Enterprise Explorer, right-click on *lab11EJB/lab11EJB/Session Beans/ShippingBean* and choose *Run As - Run on Server* to start the wizard.

b. On the first wizard page, ensure that the WebSphere Application Server v8 entry is selected, then press Finish.

RAD will start the server if necessary; this will take some time so please be patient.

Once the server starts, RAD will start the *Universal Test Client*, which lets you test your EJB.

c. In the UTC's left-hand navigation pane, expand the *EJB Beans/ShippingBeanRemote* entry until you see the *double calcDomestic(String)* entry.

d. Click on the *double calcDomestic(String)* entry: the UTC displays a form into which you can enter the method's parameter. Enter **54321** then press Invoke. The UTC creates an instance of your EJB, runs the calcDomestic method and returns the result. While this is not very exciting, this lets you test the basic function of your EJB, saving you the bother of writing an actual client program.

You can close the UTC when you are finished.

_7. Update the *ship/ShippingRemote.java* interface, adding the following method definition:

```
public double calculateInternational(String country);
```

_8. Complete the *ship/ShippingBean.java* implementation and expose it as a Web service:

a. Write an implementation for the calculateInternational method. Return any value you please.

b. Add the **@WebService** annotation to the class. Note that you may have a warning in the service indicating that since there's no "router project", the EJB is inaccessible as a Web service. You will fix that next.

_9. Next, you must expose the EJB so it can respond to SOAP requests over HTTP:

a. In the Enterprise Explorer, expand the *lab11EJB* project and its nested *Services* folder.

b. In Services, right-click on the *{http://ship}ShippingBeanService* entry and choose *Create Router Modules (EndpointEnabler)* to start the wizard.

Copyright © Descriptor Systems 2014. All Rights Reserved. Materials may not be reproduced in any form without prior written consent of Joel Barnum.

c. On the first wizard page, ensure that only the *HTTP* box in the *EJB Web Service Binding* section is checked, then press Finish.

   The warning in the service should now be gone. Note that if you have errors in the "router" project complaining about deployment descriptors, those are benign and you can ignore them.

d. Save all files and republish the project on the server.

_10. As you have done in earlier labs, test the service using the Web Services Explorer.

Part 2: Writing a Web Service Client

In this part, you will write a simple client for your shipping-calculation Web service.

Steps:

_1. As you have done in earlier labs, create either a standalone Java client or a servlet-based client for the "shipping" Web service.

Copyright © Descriptor Systems 2014. All Rights Reserved. Materials may not be reproduced in any form without prior written consent of Joel Barnum.

Copyright © Descriptor Systems 2014. All Rights Reserved. Materials may not be reproduced in any form without prior written consent of Joel Barnum.