


How-To Build and Use Web Services with JDeveloper

Web Services provide client neutral access to data and other services. JDeveloper allows you to create different types of Web Services quickly and easily....

In this tutorial, you will create 4 different Web Services: a POJO Annotation-Driven service, a Declaratively-Driven POJO service, a service for existing WSDL, and an EJB service. The focus of these scenarios is to demonstrate and test Java EE web services. In particular this means JAX-WS (Java API for XML Web Services) and annotation handling. JAX-WS enables you to enter annotations directly into the Java source without the need for a separate XML deployment descriptor.

At the end of the tutorial you create an ADF Client application that consumes the web services you created.

Purpose	Duration	Application
<p>This tutorial shows you how to build and consume Web Services. The tutorial shows several end-to-end scenarios for creating web services. After you develop several web services, you create a client application that uses those services.</p> <p>To see the complete application you will create, click the Download button to download a zip of the solution application, and then unzip it in a workspace folder of your choice.</p>	4 hours	

Part 1: Building a POJO Annotation-Driven Service

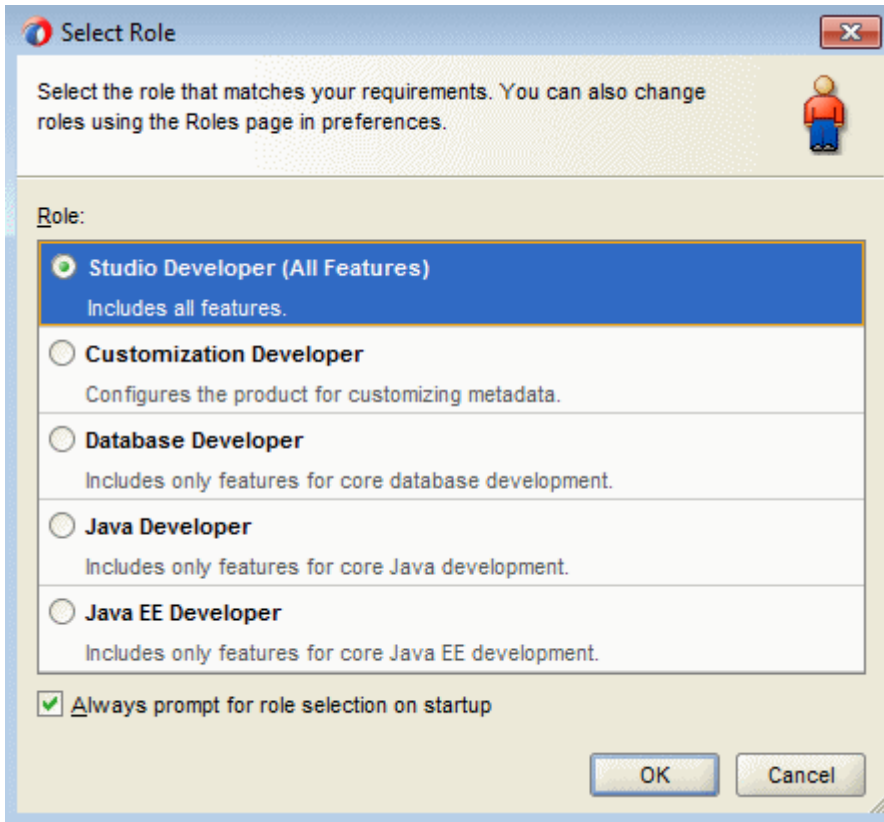
In this first part of the tutorial, you install the required lab files, start JDeveloper, and open the startup application and project.

Step 1: Getting Ready

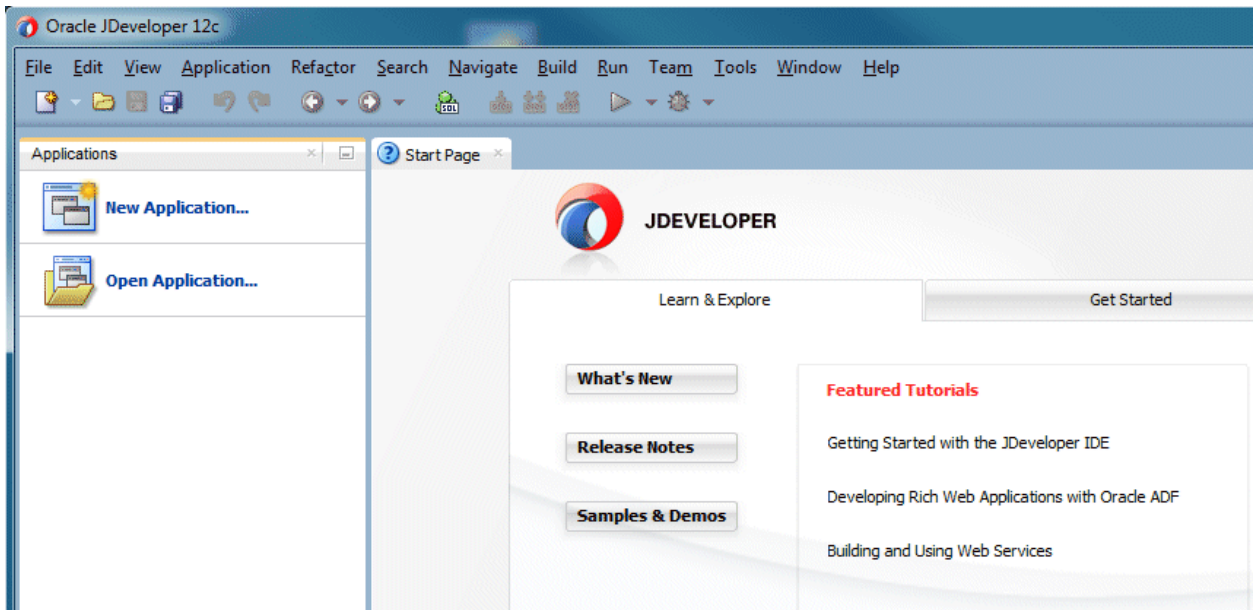
1. Download the [lab starter files](#) and save the webservice.zip file in a temporary folder (such as d:\Temp.)
2. Using WinZip or whatever zip utility you have, unzip the webservice.zip into a folder of your choice. In this tutorial, we used C:\JDeveloper\mywork.
3. Start JDeveloper by selecting **Start > Programs > <JDEVELOPER_HOME> > OracleHome > Oracle JDeveloper Studio > Oracle JDeveloper Studio**

If a dialog box opens asking if you would like to import preferences from a previous JDeveloper installation, click **NO**.

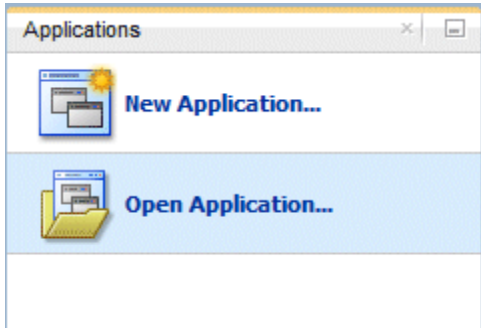
4. If prompted for a Role, select **Studio Developer**.



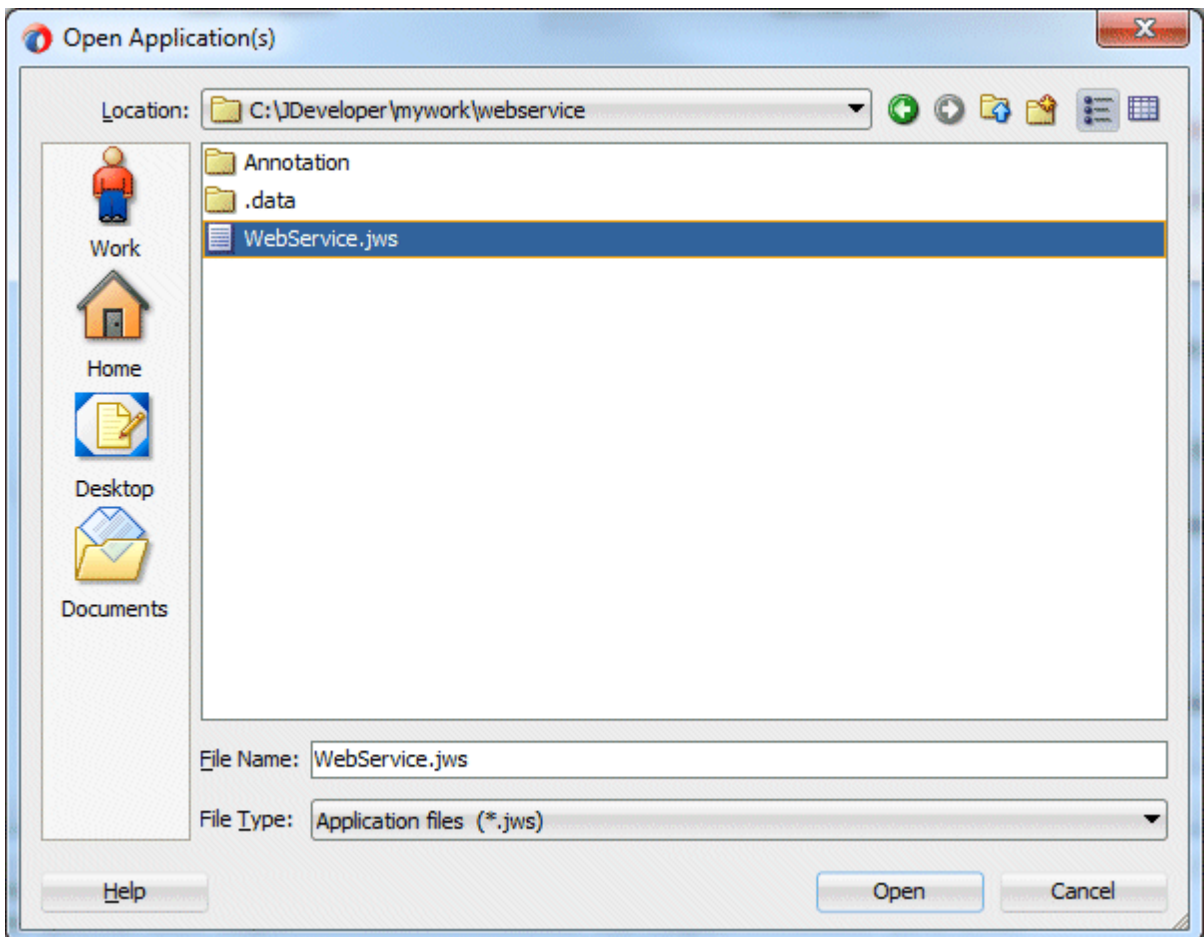
5. If the **Tip of the Day** window opens, click **Close**.
6. You should now see the JDeveloper IDE. Close the **Start page** by hovering your mouse over the tab and clicking the **X** on the tab.



7. Select the **Applications** window tab and click **Open Application** (alternatively, you can select **File > Open**)

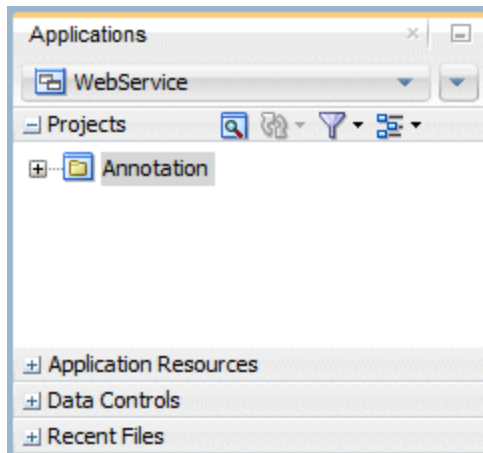


8. In the **Open Application** dialog box, locate the **Web Service** folder where you unzipped the WebService.zip file and select **WebService.jws**.



9. Click **Open**
10. If you are prompted to migrate the application, click **Yes**.

The Applications window should look like this:

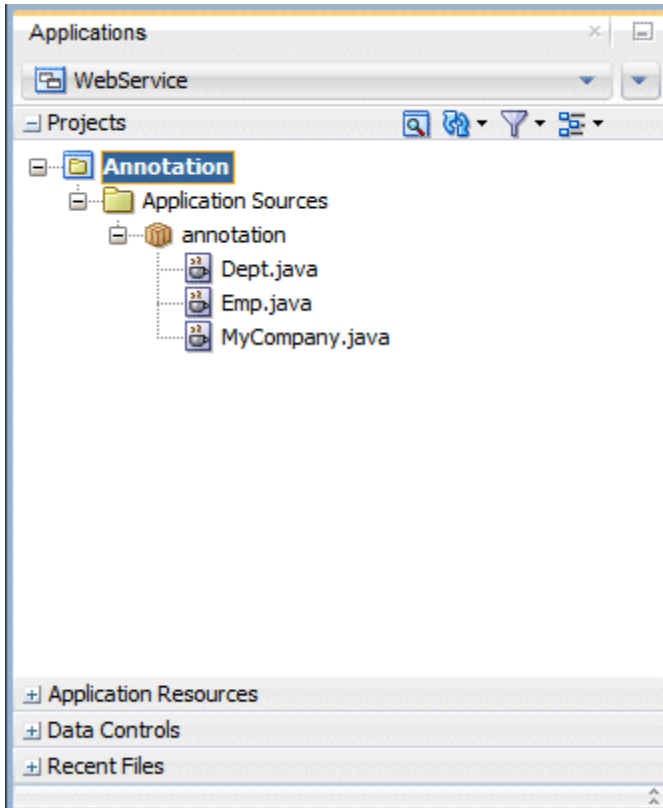


Step 2: Adding a Plain Old Java Object (POJO) to contain a Web Service Method

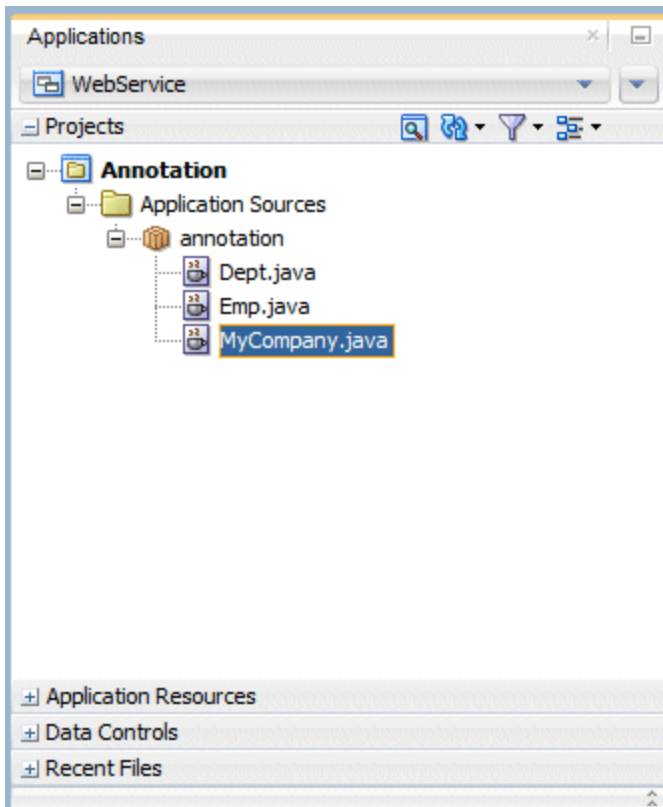
In this section you start with a project that contains plain old Java classes and add an annotated method that you publish as a web service.

Web service annotation is a feature of Java EE 6 which takes complexity out of creating and deploying Web Services. Web service annotation allows you to define web services from within a POJO. This feature of Java EE eliminates the need for complex configuration of the web service and the web server. Java EE introspects the deployed classes and creates the web server configuration on-the-fly. This frees up the developer to concentrate more on the service rather than the tedious details of deployment.

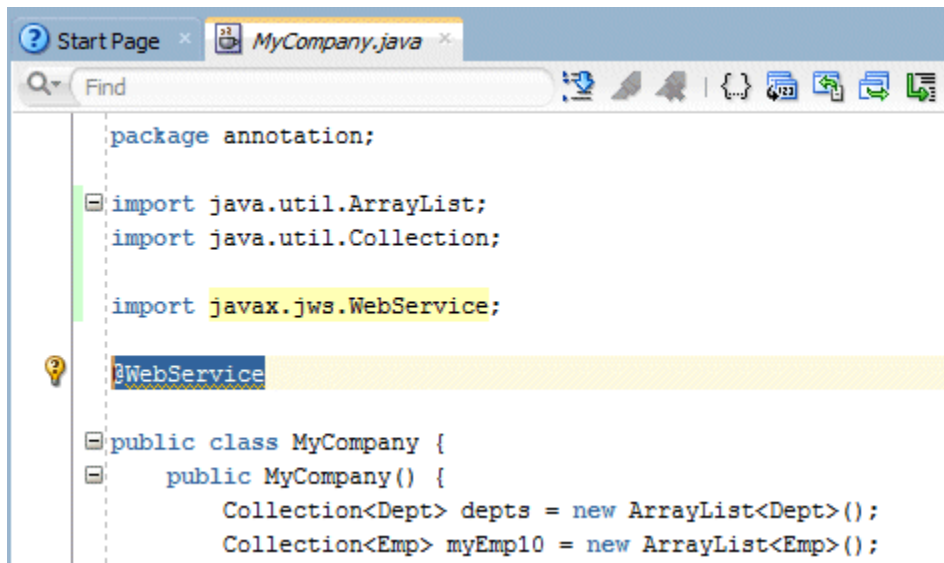
1. In the Applications window, expand the **Annotation** project nodes to show the POJO classes:
 - Dept.java describes the department structure
 - Emp.java describes the employee structure
 - MyCompany.java populates information about departments and employees



2. In the Applications window, double-click **MyCompany.java** to edit it.



3. Add an `@WebService` annotation after the import statements. The IDE will prompt you to select the import for the `WebService` class. Select `javax.jws.WebService` from the popup. This annotation denotes that the class contains a method to be used by a web service.



```
package annotation;

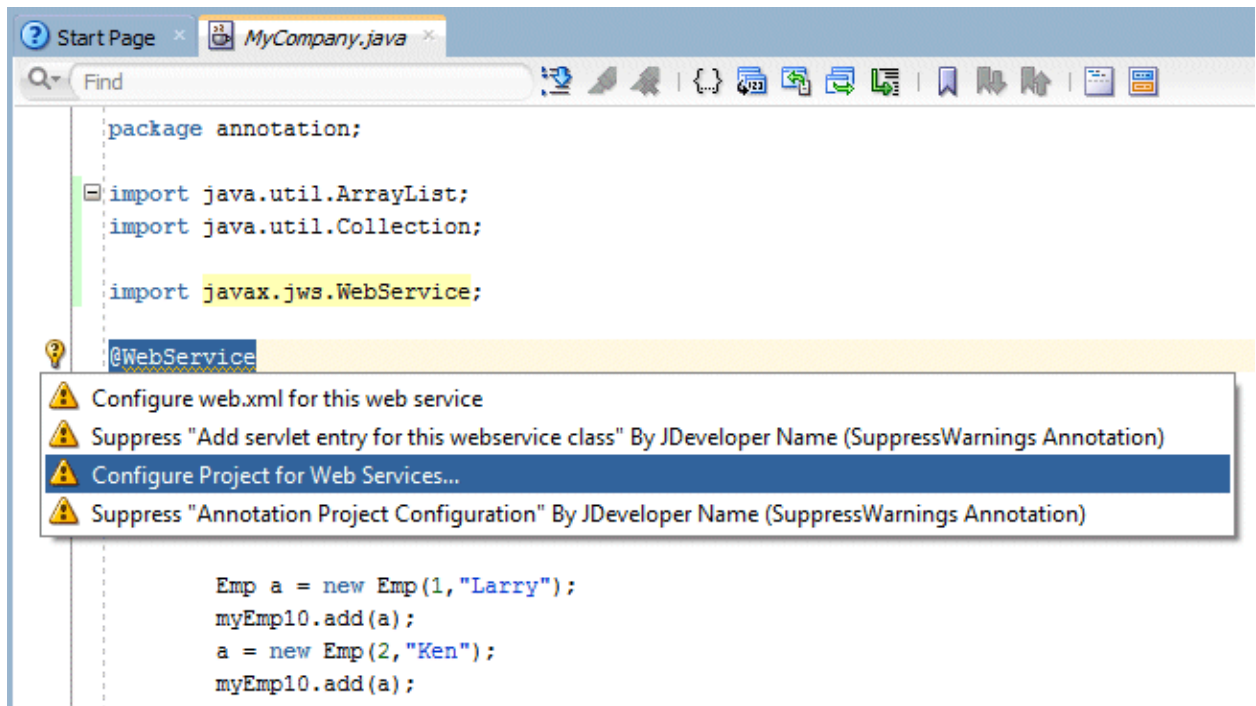
import java.util.ArrayList;
import java.util.Collection;

import javax.jws.WebService;

@WebService

public class MyCompany {
    public MyCompany() {
        Collection<Dept> depts = new ArrayList<Dept>();
        Collection<Emp> myEmp10 = new ArrayList<Emp>();
```

4. In the margin of the editor, click **Quick Hint** (light bulb icon) and select the **Configure project for web services** option.



```
package annotation;

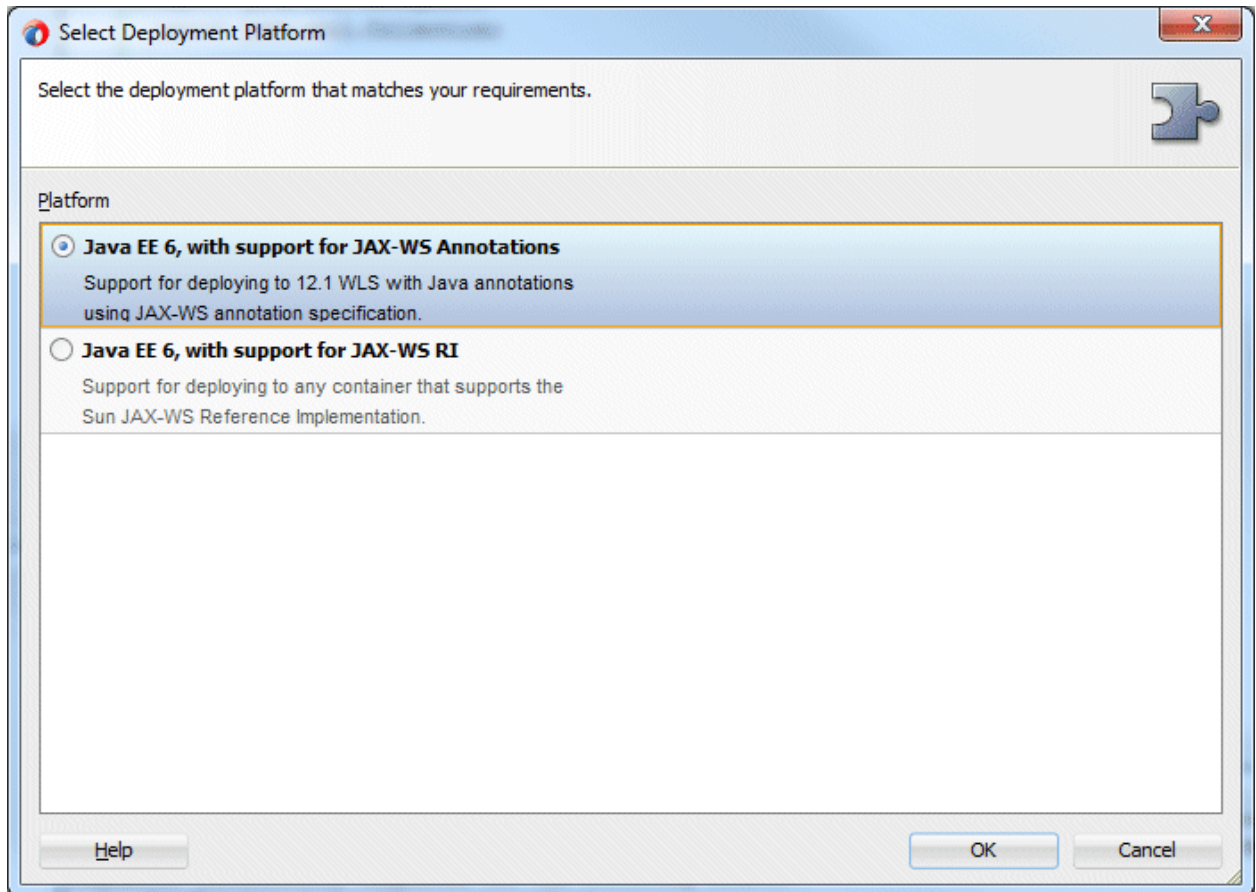
import java.util.ArrayList;
import java.util.Collection;

import javax.jws.WebService;

@WebService

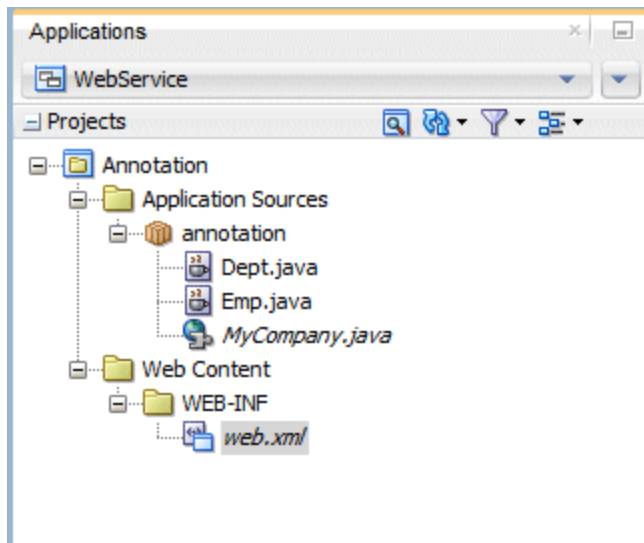
Emp a = new Emp(1, "Larry");
myEmp10.add(a);
a = new Emp(2, "Ken");
myEmp10.add(a);
```


5. In the **Select Deployment Platform** dialog box, ensure that **Java EE 6, with support for JAX-WS Annotations** is selected.



6. Click **OK**. This step adds the `javax.jws.WebService` import statement to the Java class if it is not already there and creates a `web.xml` file.
The Applications window should look like the following:

Notice that the icon for `MyCompany.java` class is changed to represent a `WebService` class, and the `web.xml` file has been added to your project.



7. Click **Save All**  to save your work.
8. In the Code Editor, scroll to the bottom of the class and add the following code statements:

```
public Dept getDeptInfo (int id) {  
    for (Dept a: this.getMyDepts() ) {  
        if (a.getId() == id) {  
            return a;  
        }  
    }  
    return null;  
}
```

This loop returns information about all employees working in a specific department. The code in the editor window should look like:


```
public boolean addEmployeeToDept(Emp emp, int deptid){
    //TODO write some logic here
    System.out.println("Here we'll be adding an employee to " +deptid);
    return true;
}

public Dept getDeptInfo (int id) {
    for (Dept a: this. getMyDepts() ) {
        if (a.getId() == id) {
            return a;
        }
    }
    return null;
}
}
```

MyCompany
Source History

9. Create a second annotation before the **getDeptInfo()** method. The annotation signifies this is the method to be exposed from the web service. Add a blank line above the **getDeptInfo()** method, and start typing **@WebMethod**. Code insight pops up a list of available syntaxes. Select **WebMethod** from the list.

```
public Collection<Dept> getMyDepts() {
    return myDepts;
}

public boolean addEmployeeToDept(Emp emp, int deptid){
    //TODO write some logic here
    System.out.println("Here we'll be adding an employee to " +deptid);
    return true;
}

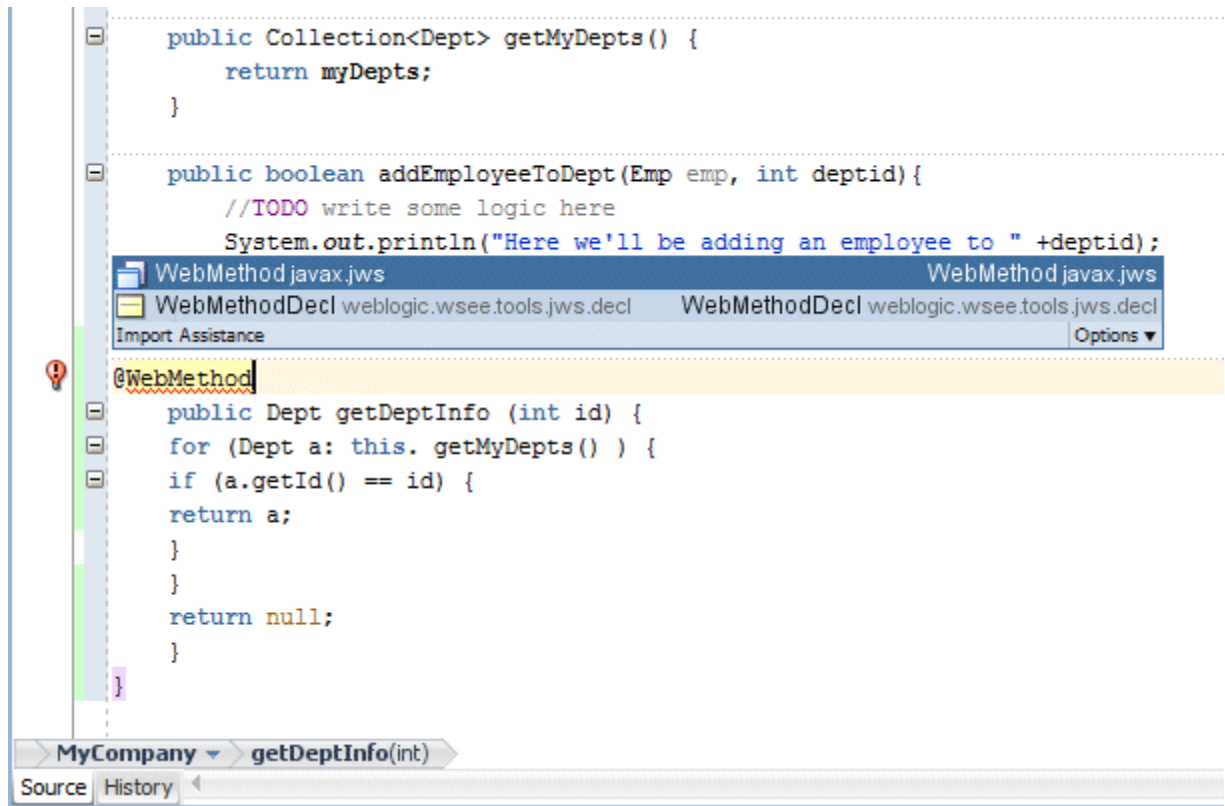
@WebMethod
public Dept getDeptInfo (int id) {
    for (Dept a: this. getMyDepts() ) {
        if (a.getId() == id) {
            return a;
        }
    }
    return null;
}
}
```

Select import for WebMethod... (Alt-Enter)

MyCompany getDeptInfo(int)
Source History

10. If suggested, press [Alt]+[Enter] to add the `import javax.jws.WebMethod;` statement (although this statement may be added automatically.)

The class should now look like the following:




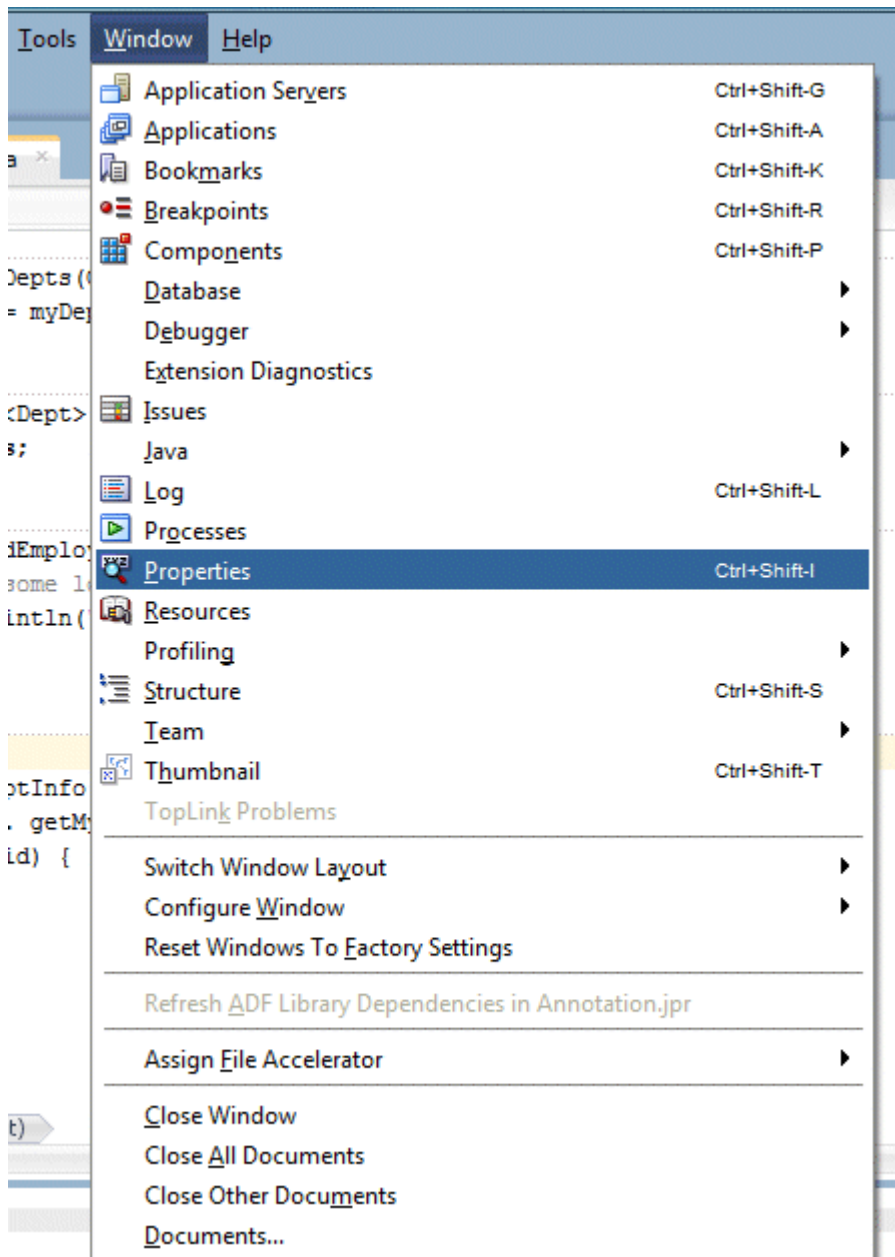
```
public Collection<Dept> getMyDepts() {
    return myDepts;
}

public boolean addEmployeeToDept(Emp emp, int deptid) {
    //TODO write some logic here
    System.out.println("Here we'll be adding an employee to " +deptid);
}

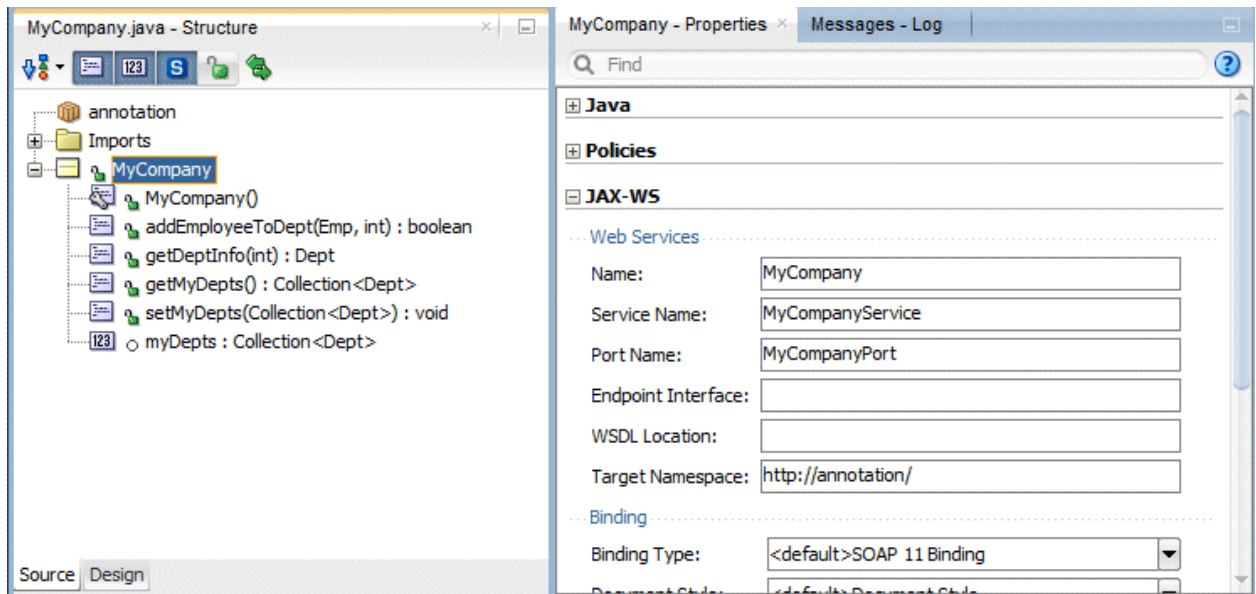
@WebMethod
public Dept getDeptInfo (int id) {
    for (Dept a: this. getMyDepts() ) {
        if (a.getId() == id) {
            return a;
        }
    }
    return null;
}
```

The screenshot shows an IDE with a code editor containing the above Java code. An import assistance popup is visible over the `@WebMethod` annotation, listing `WebMethod javax.jws` and `WebMethodDecl weblogic.wsee.tools.jws.decl`. The IDE's breadcrumb at the bottom shows `MyCompany > getDeptInfo(int)`.

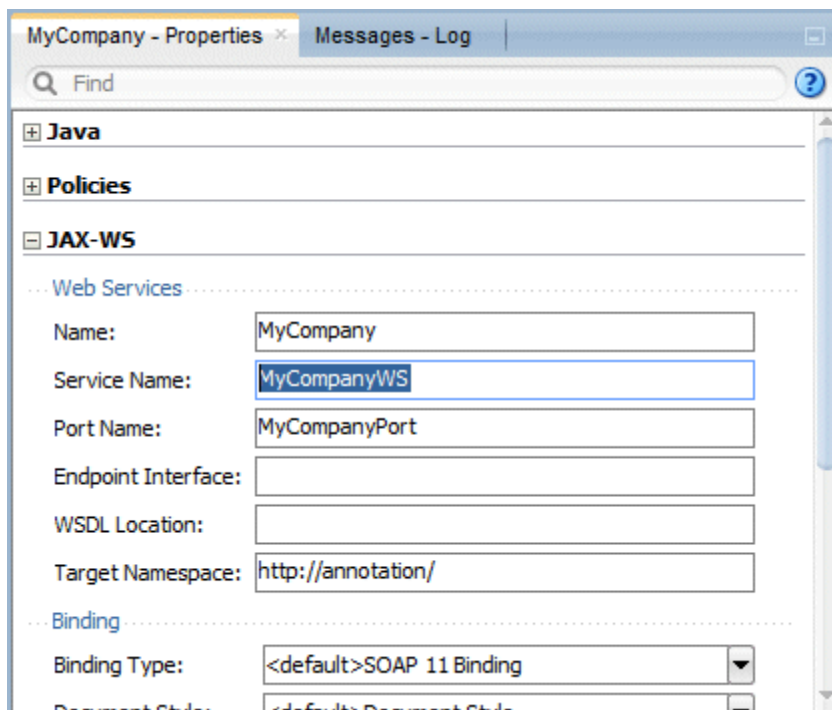
11. Click **Save All**  to save your work.
12. You can use the Properties window to modify the characteristics of the class. In the menu bar, select **Window > Properties** and it will open as a tab in the bottom portion of the IDE. Note: If the Properties window opens in a different part of the IDE, you can drag its tab and drop it on the bottom panel if you would rather work with it there.



13. To display the properties of the MyCompany class in the **Properties** window, select the **Source** tab at the bottom of the Structure window, then select the top level **MyCompany** class name.



14. The Properties window displays a few expandable nodes. Expand the **JAX-WS** node and notice that the **Service Name** has the word '**Service**' appended to the class name.
15. Change the **Service Name** to **MyCompanyWS**. Notice that the class reflects the name change.



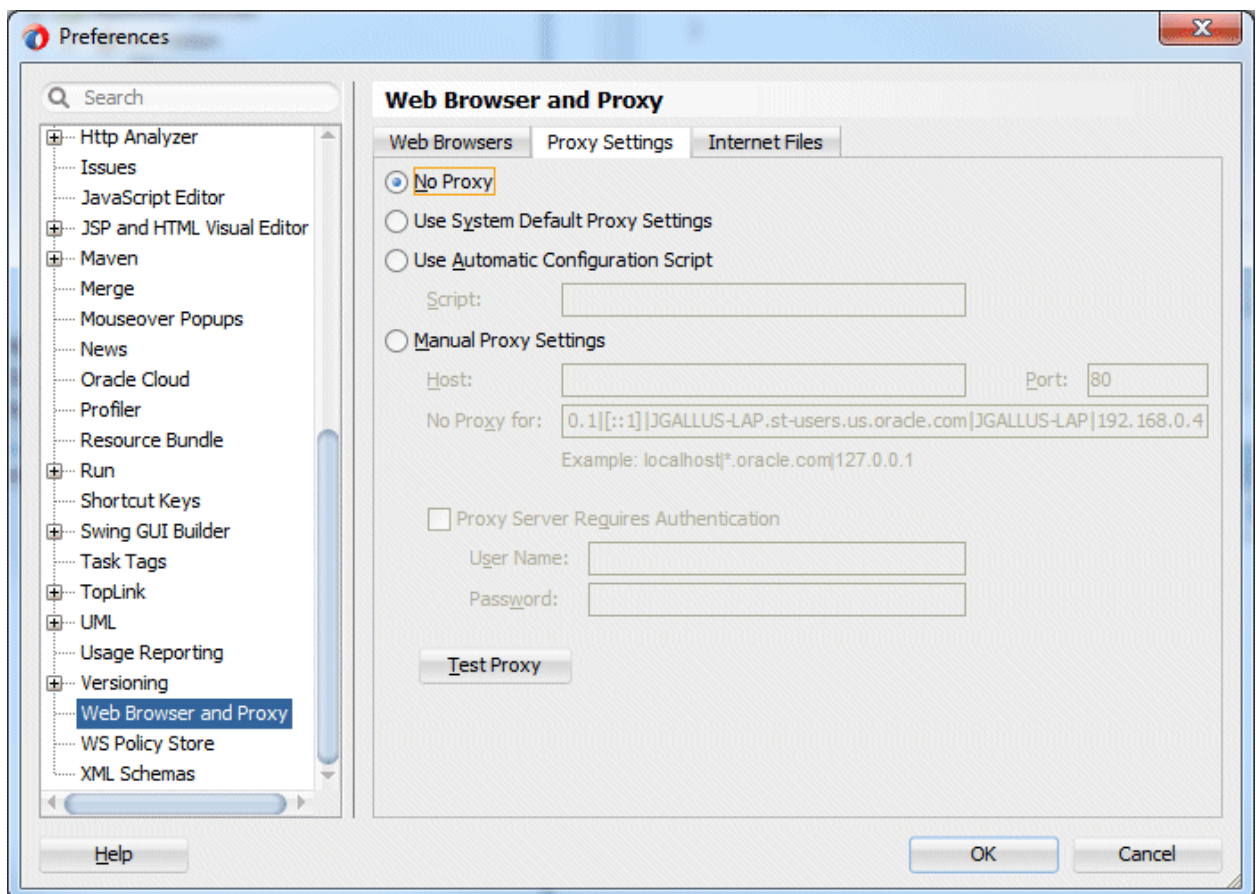
16. Click **Save All**  to save your work.

You have now created a POJO Web Service. In this next section, you will test you Web Service.

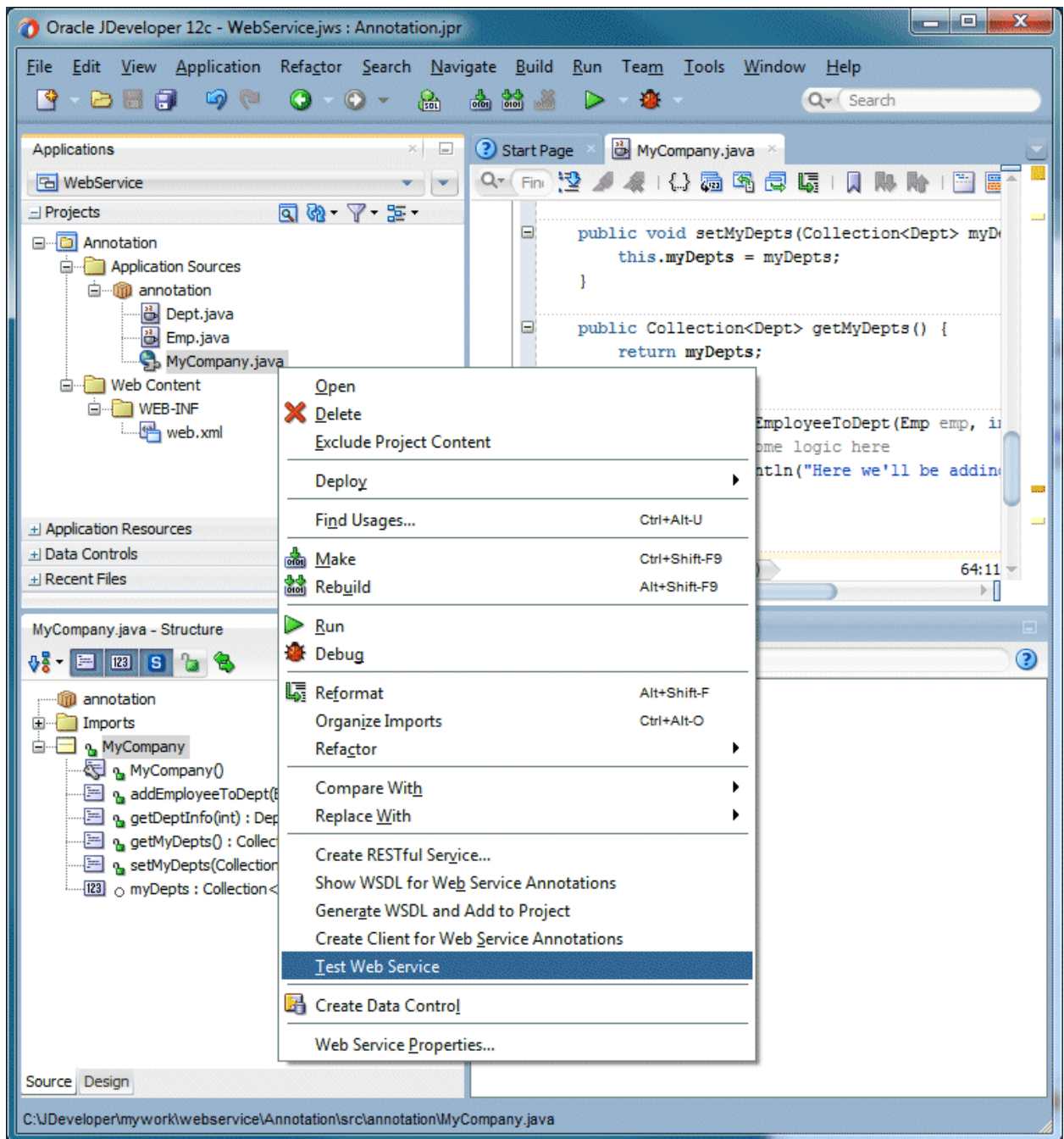
Step 3: Testing a Web Service

In this section you compile, deploy and test the web service using the HTTP Analyzer. JDeveloper includes a web service testing mechanism called the HTTP Analyzer. When you use the HTTP analyzer to test web services, JDeveloper compiles and deploys the service to the integrated web server. It then invokes the analyzer, allowing you to send and receive values from the web service.

1. Before testing the web service, check that your web browser settings are correct. Select **Tools > Preferences** and then scroll down the list on the left to select the **Web Browser and Proxy** page. On the Proxy Settings tab, ensure that the **No Proxy** is selected, then click **OK**.

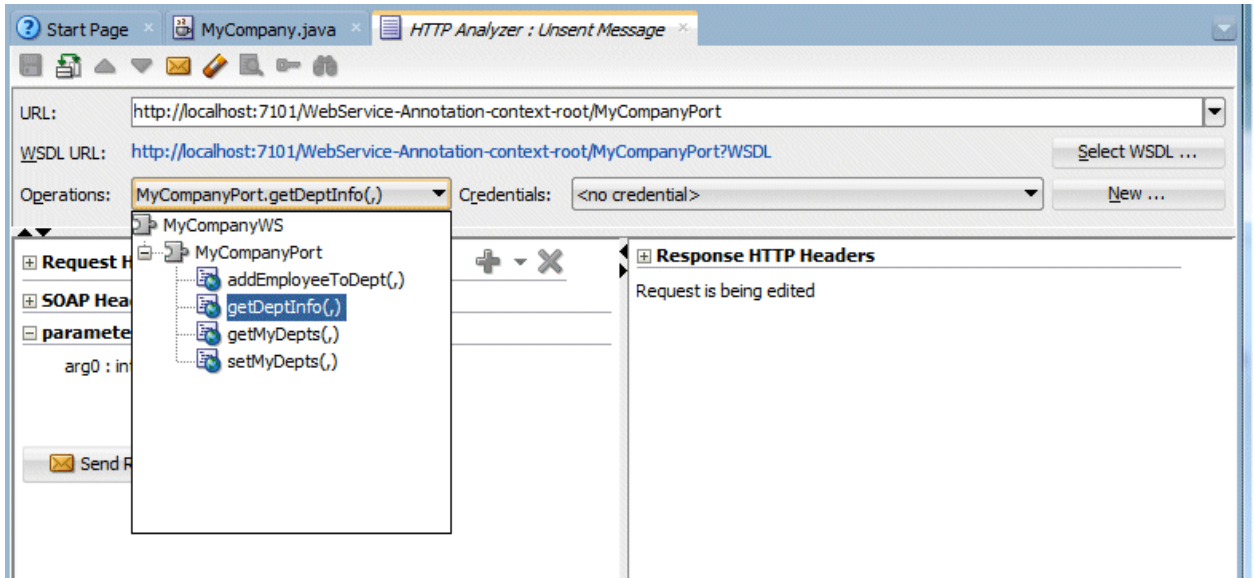


2. In the Applications window, **right-click** the MyCompany.java node and in the context menu, select **Test Web Service**.

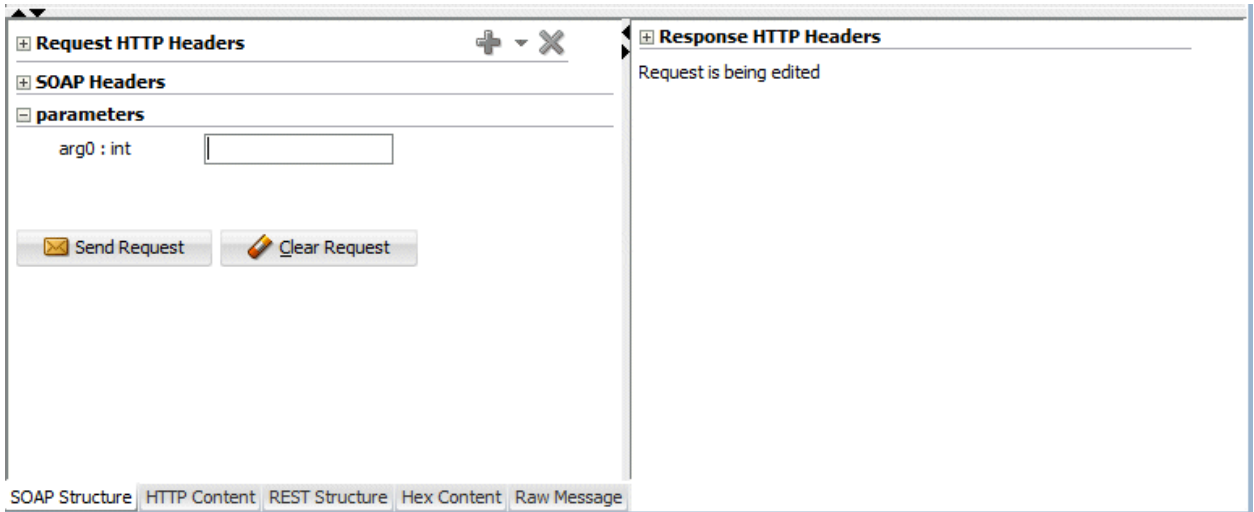


This option invokes the integrated WebLogic Server, deploys the service, and then starts the analyzer. It may take a few seconds to start WebLogic Server if you are running it for the first time. If this is the first time you test a service, Windows may ask you about blocking content. Allow the content to be displayed.

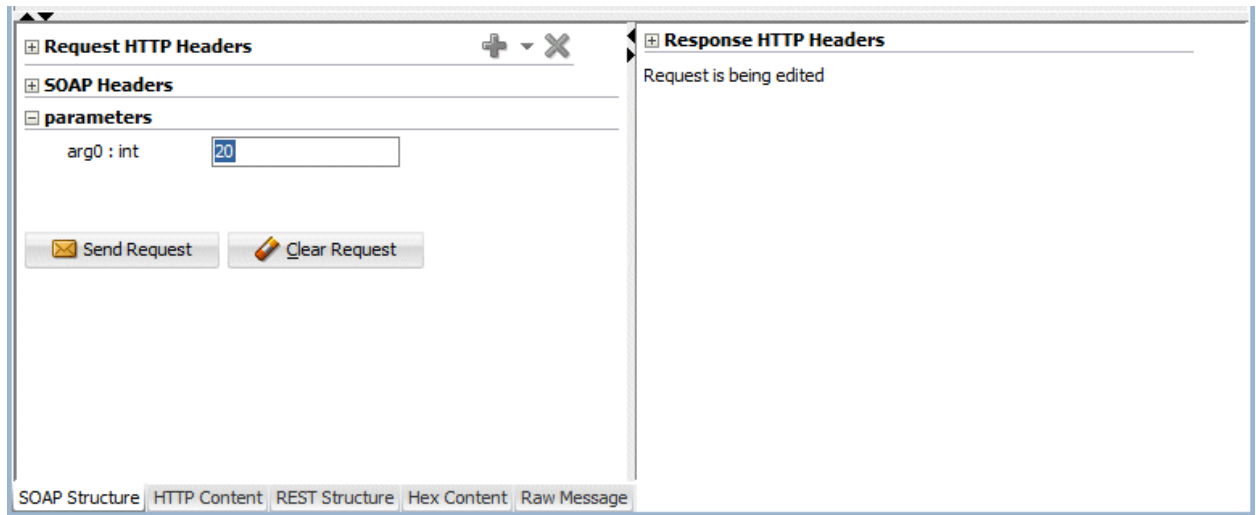
3. The top portion of the **HTTP Analyzer** editor window displays the URL for the web service, the WSDL URL, and the exposed Operations. Select the `MyCompanyPort.getDeptInfo(.)` operation from the list.




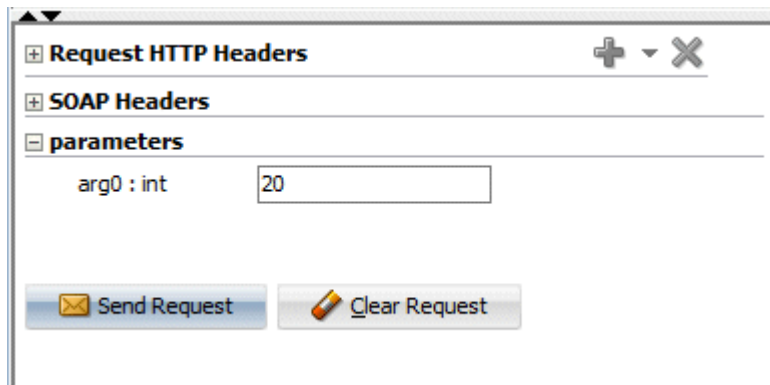
The bottom portion of the analyzer is split into two areas: Request and Response. The request area shows all the arguments from the exposed method (in this case, only one argument.) When the web service is executed, the Response area shows the results.



4. In the Request area, enter a department number value (10, 20 or 30) in the **arg0** field.



5. In the toolbar area of the analyzer, click **Send Request**, or click the Send Request button  below the argument.



6. The analyzer sends the request to the service, returning after a few seconds the information about employees working in the specified department.

The screenshot shows a SOAP client interface with two main panels: "Request HTTP Headers" and "Response HTTP Headers".

Request HTTP Headers: Shows "SOAP Headers" with a "parameters" section containing "arg0 : int" with the value "20". There are "Send Request" and "Clear Request" buttons.

Response HTTP Headers: Shows a "200 OK" status. The "parameters" section includes a "return" object with an "employees : Array". The array contains two employee objects:

Employee	id	name	salary
1	3	Gary	0.0
2	4	Jeff	0.0

At the bottom, there are tabs for "SOAP Structure", "HTTP Content", "REST Structure", "Hex Content", and "Raw Message".

7. Click the **HTTP Content** tab at the bottom of the editor to look at the xml code.

The screenshot shows the same SOAP client interface, but with the "HTTP Content" tab selected at the bottom. The main area displays the XML code for both the request and response.

Request XML:

```
<?xml version = '1.0' encoding = 'UTF-8'?>
<env:Envelope xmlns:env="http://schemas.xmlsoap.org/soap/envelope/">
  <env:Header/>
  <env:Body>
    <ns1:getDeptInfo>
      <arg0>20</arg0>
    </ns1:getDeptInfo>
  </env:Body>
</env:Envelope>
```

Response XML:

```
<?xml version = '1.0' encoding = 'UTF-8'?>
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
  <S:Body>
    <ns0:getDeptInfoResponse xmlns:ns0="http://schemas.xmlsoap.org/soap/envelope/">
      <return>
        <employees>
          <id>3</id>
          <name>Gary</name>
          <salary>0.0</salary>
        </employees>
        <employees>
          <id>4</id>
          <name>Jeff</name>
          <salary>0.0</salary>
        </employees>
      </return>
    </ns0:getDeptInfoResponse>
  </S:Body>
</S:Envelope>
```

At the bottom, there are tabs for "SOAP Structure", "HTTP Content", "REST Structure", "Hex Content", and "Raw Message".

8. Click the **Raw Message** tab at the bottom of the editor for another presentation of the code.

```
POST http://localhost:7101/WebService-Annotation-co
SOAPAction: ""
Content-Type: text/xml; charset=UTF-8
Host: localhost:7101
Content-Length: 198
X-HTTPAnalyzer-Rules: 1@localhost:8099

<?xml version = '1.0' encoding = 'UTF-8'?>
<env:Envelope xmlns:env="http://schemas.xmlsoap.org
  <env:Header/>
  <env:Body>
    <ns1:getDeptInfo>
      <arg0>20</arg0>
    </ns1:getDeptInfo>
  </env:Body>
</env:Envelope>

HTTP/1.1 200 OK
Content-Type: text/xml; charset=utf-8
Date: Thu, 22 May 2014 20:08:28 GMT
X-ORACLE-DMS-ECID: 12877853-c418-4894-9574-8a9df878
Content-Length: 512
X-HTTPAnalyzer-RuleName: Pass through :

<?xml version = '1.0' encoding = 'UTF-8'?>
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soa
  <S:Body>
    <ns0:getDeptInfoResponse xmlns:ns0="http://an
      <return>
        <employees>
          <id>3</id>
          <name>Gary</name>
          <salary>0.0</salary>
        </employees>
```

9. Click the **SOAP Structure** tab at the bottom of the editor, and then in the top part of the HTTP Analyzer, click the **WSDL URL** link.

URL: <http://localhost:7101/WebService-Annotation-context-root/MyCompanyPort>

WSDL URL: <http://localhost:7101/WebService-Annotation-context-root/MyCompanyPort?WSDL>

Operations: MyCo <http://localhost:7101/WebService-Annotation-context-root/MyCompanyPort?WSDL>

Request HTTP Headers

SOAP Headers

parameters

arg0 : int

Response HTTP Headers 200 OK

parameters

return

employees : Array

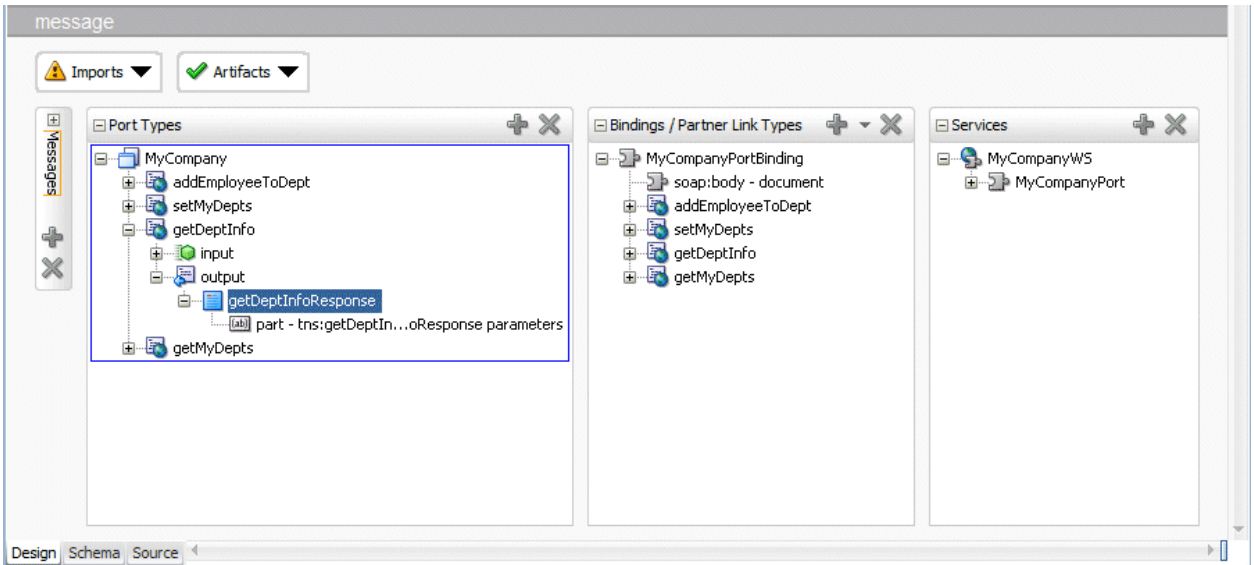
employees

id	3
name	Gary
salary	0.0

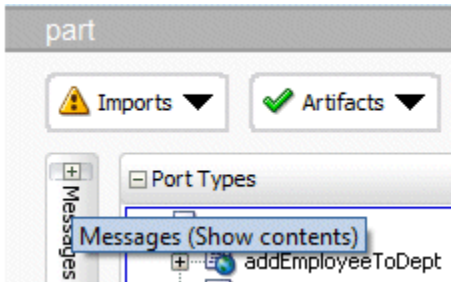
employees

id	4
name	Jeff
salary	0.0

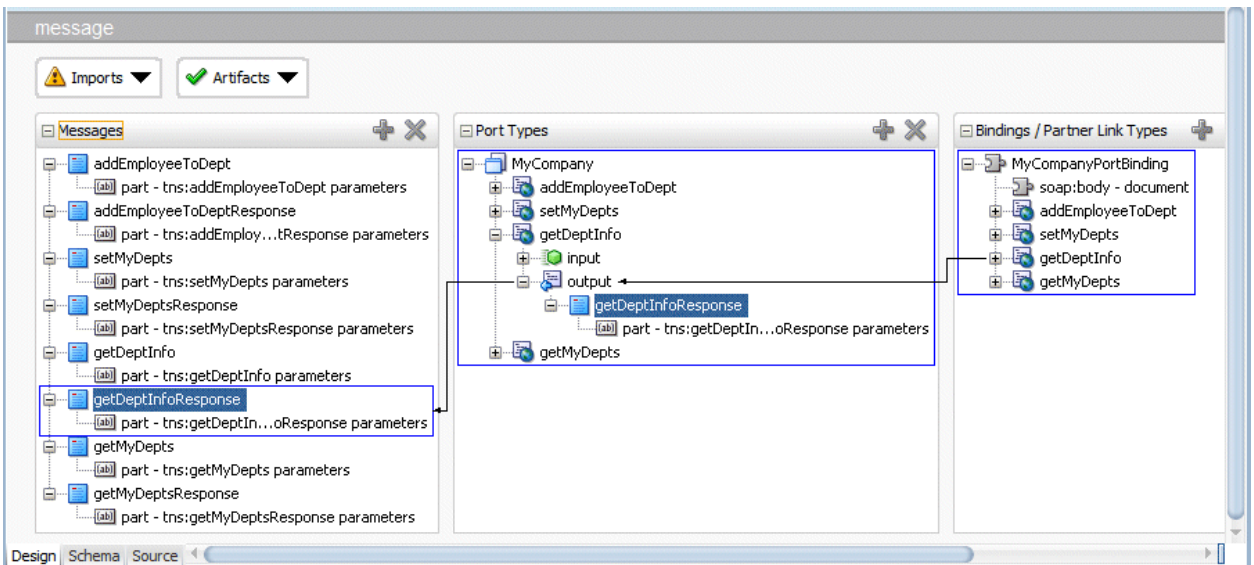
10. This opens the visual editor for the web service. In the Port Types panel, expand the **getDeptInfo** > **output** > **getDeptInfoResponse** nodes.



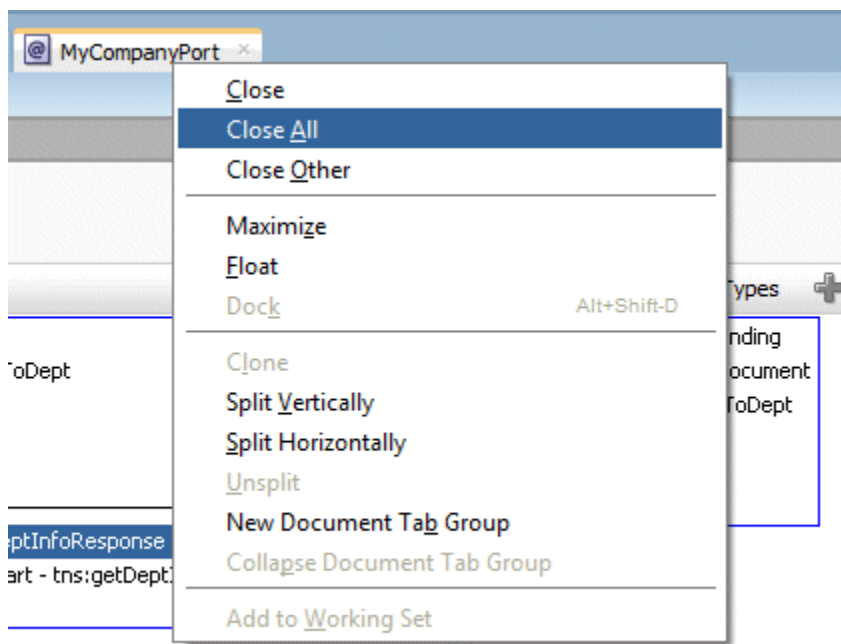
- To the left of the Port Types panel, click the small **Plus** sign at the top of **Messages** to show message contents.



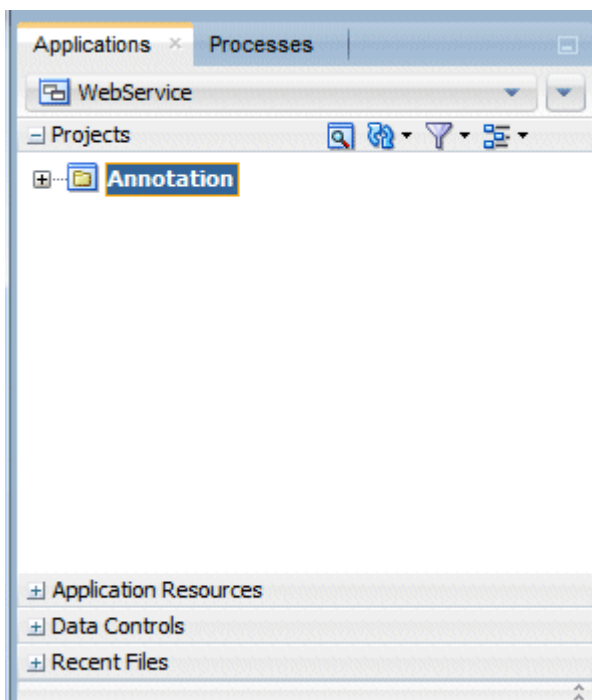
A new graphical representation shows the flow for any message you select.



12. Right-click any tab in the editor window and select the **Close All** option.



13. Collapse the **Annotation** project node in the Applications window.



Courtesy: https://docs.oracle.com/cd/E53569_01/tutorials/tut_web_services/tut_web_services.html

Modified: 2021.10.14.7.34.PM

Dököll Solutions, Inc.