WA2018 Programming REST Web Services with JAX-RS 1.1 - WebLogic 12c / Eclipse

Student Labs

Web Age Solutions Inc.

Table of Contents

Lab 1 - Configure the Development Environment

Throughout these labs, you will be developing web service components with two main pieces of software: Oracle WebLogic Server v12c and Oracle Enterprise Pack for Eclipse(OEPE).

Oracle WebLogic Server v12c(12.1.1) (WLS) is a JEE application server that serves up JEE applications, and is also a web service container.

OEPE, Provides tools that make it easier to develop applications utilizing specific Oracle Fusion Middleware technologies and Oracle Database. Based on the Eclipse tool, it is the programming environment we will use for this class.

In this lab exercise, we will configure WLS and OEPE to form our development environment.

Part 1 - Oracle WebLogic server Setup

WebLogic Server is already extracted into **C:\Software\WebLogic** folder.

In this part we will setup necessary Environment variables and run installation configuration script.

Follow the instructions to set up JAVA_HOME, MW_HOME and JAVA_VENDOR environment variable

JAVA_HOME=C:\Software\Java

MW_HOME=C:\Software\WebLogic

JAVA_VENDOR=Oracle

__1. Right click on the **My Computer** icon on your computer and select **properties**.

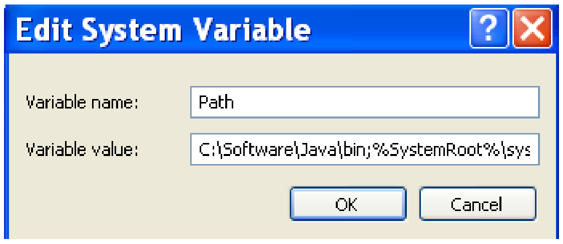
__2. Click the **Advanced** Tab.

__3. Click the **Environment Variables** button.

__4. From the *System Variables* list, select **Path** and click **Edit**.

__5. At the beginning of the line enter the following:

```
C:\Software\Java\bin;
```



Variable name: Path

Variable value: C:\Software\Java\bin;%SystemRoot%\sys

__6. Click **OK**.

__7. Under System Variables, click **New**.

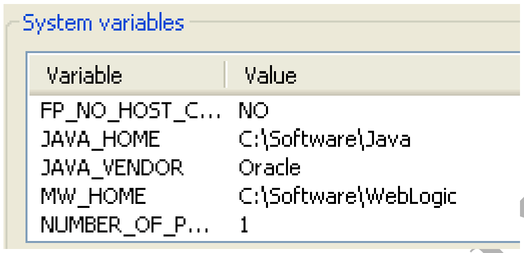
__8. Enter the variable name as **JAVA_HOME**

__9. Enter the variable value as the install path for the Development Kit.

__10. Click **OK**.

__11. Repeat these steps for **MW_HOME** and **JAVA_VENDOR** variables.

Once all done it should look like.



System variables

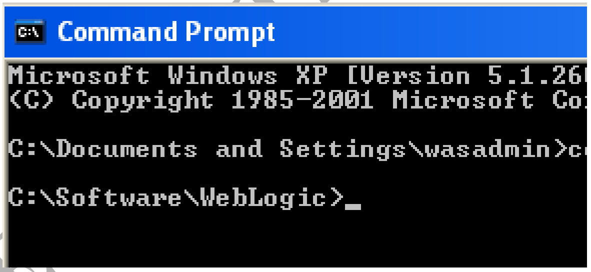
Variable	Value
FP_NO_HOST_C...	NO
JAVA_HOME	C:\Software\Java
JAVA_VENDOR	Oracle
MW_HOME	C:\Software\WebLogic
NUMBER_OF_P...	1

__12. Click OK until you close all windows.

Now you will run the installation configuration script in the MW_HOME directory. It should be run only once.

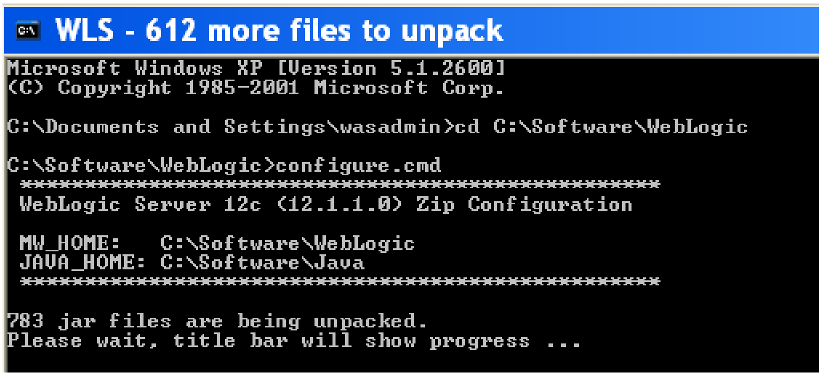
__13. Open a command prompt.

__14. Change the directory to **C:\Software\WebLogic**



```
Microsoft Windows XP [Version 5.1.26
(C) Copyright 1985-2001 Microsoft Co

C:\Documents and Settings\wasadmin>c

C:\Software\WebLogic>_
```

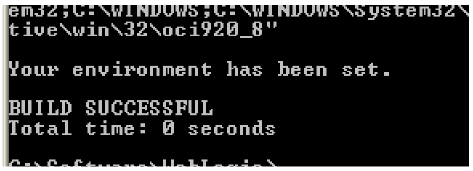
__15. Enter **configure.cmd** and hit Enter.



```
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\wasadmin>cd C:\Software\WebLogic

C:\Software\WebLogic>configure.cmd
 **************************************************
 WebLogic Server 12c (12.1.1.0) Zip Configuration

 MW_HOME:   C:\Software\WebLogic
 JAVA_HOME: C:\Software\Java
 **************************************************

783 jar files are being unpacked.
Please wait, title bar will show progress ...
```

It takes few minutes to complete.

You will get a notification when the installation is done.



```
em32;C:\WINDOWS;C:\WINDOWS\System32\
tive\win\32\oci920_8"

Your environment has been set.

BUILD SUCCESSFUL
Total time: 0 seconds

C:\Software\WebLogic>
```

__16. Close the command prompt window.

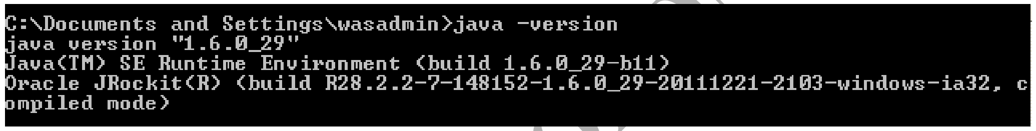
__17. Restart the machine.

__1. Open a Windows command prompt. You can do this by selecting '**Start -> Run**', entering '**cmd**', and then pressing the **OK** button.

__2. Enter the following command:

```
java -version
```

Make sure you see the response shown below.

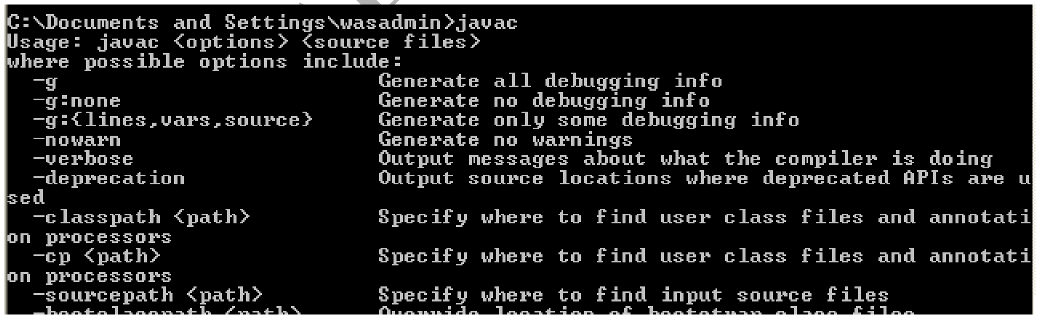


```
C:\Documents and Settings\wasadmin>java -version
java version "1.6.0_29"
Java(TM) SE Runtime Environment (build 1.6.0_29-b11)
Oracle JRockit(R) (build R28.2.2-7-148152-1.6.0_29-20111221-2103-windows-ia32, c
ompiled mode)
```

__3. Enter the following command:

```
javac
```

Verify that you get the options to run the Java compiler:



```
C:\Documents and Settings\wasadmin>javac
Usage: javac <options> <source files>
where possible options include:
  -g                         Generate all debugging info
  -g:none                    Generate no debugging info
  -g:{lines,vars,source}     Generate only some debugging info
  -nowarn                    Generate no warnings
  -verbose                   Output messages about what the compiler is doing
  -deprecation               Output source locations where deprecated APIs are u
sed
  -classpath <path>          Specify where to find user class files and annotati
on processors
  -cp <path>                 Specify where to find user class files and annotati
on processors
  -sourcepath <path>         Specify where to find input source files
  -bootclasspath <path>      Override location of bootstrap class files
```

__4. Close the command prompt window.

Part 2 - Configure a server in OEPE

We now need to configure OEPE to be able to communicate with WLS. Doing so will allow us to perform simple operations with the server (e.g. Starting, stopping and deploying applications). While this is not strictly necessary for a development environment, it significantly simplifies development time. In order to setup the server we should also need to create a domain.

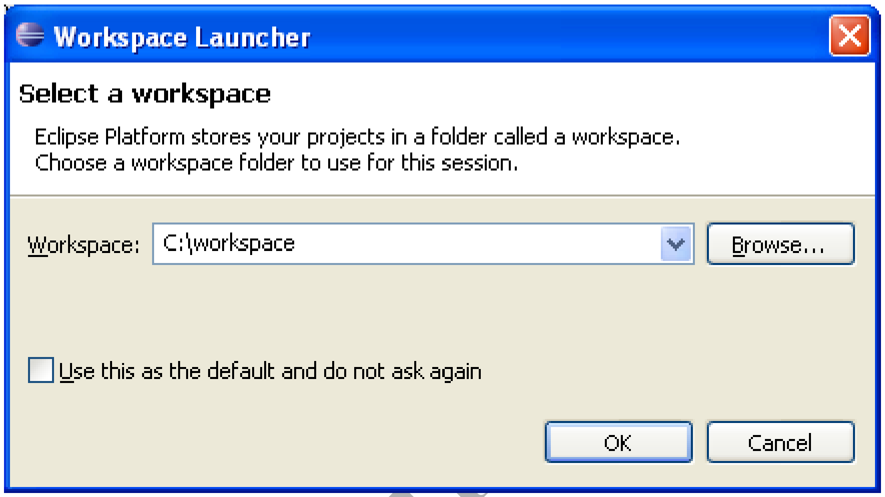
As with WLS, OEPE has already been installed on your computer. We just need to configure it.

__1. Start OEPE by launching **C:\Software\OEPE\eclipse.exe**

OEPE will launch.

You will be prompted to select a Workspace.

__2. Set the *Workspace* to **C:\workspace**



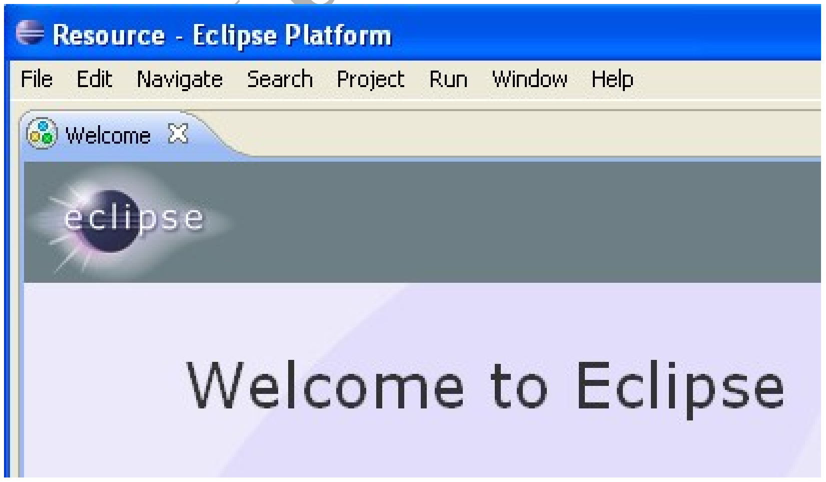
Select a workspace

Eclipse Platform stores your projects in a folder called a workspace.
Choose a workspace folder to use for this session.

Workspace: C:\workspace

__3. Click **OK**.

OEPE will start.

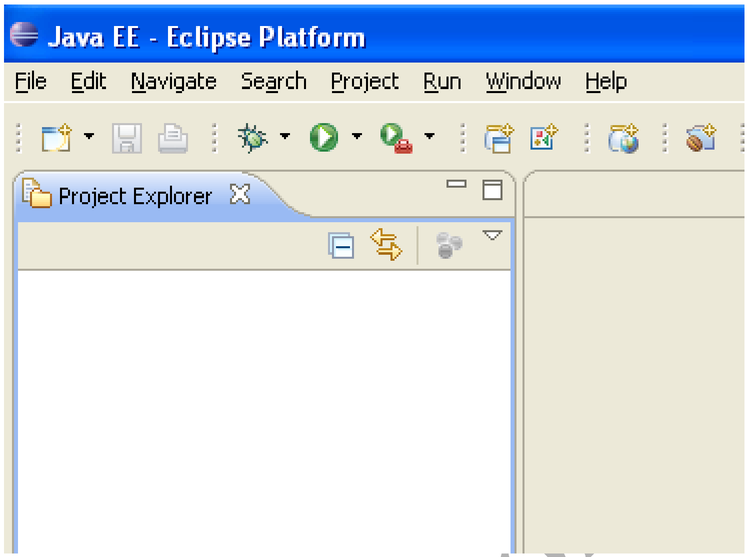


As mentioned earlier, OEPE is built on top of Eclipse. If you are familiar with Eclipse, you should have no problems using OEPE.

__4. WTP is displaying the **Welcome** screen (or *view*). Close the view by clicking the **x** in its tab.

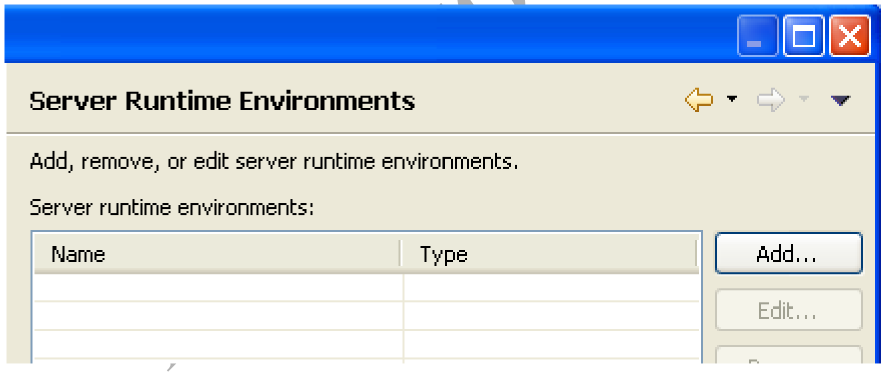
__5. From the menu, select **Window > Open Perspective > Java EE**.

OEPE will then show the **Java EE** *perspective*. (A *perspective* is a collection of views)


Java EE - Eclipse Platform
File Edit Navigate Search Project Run Window Help
Project Explorer

__6. Select **Windows | Preferences**.

__7. Expand **Server | Runtime Environments**.

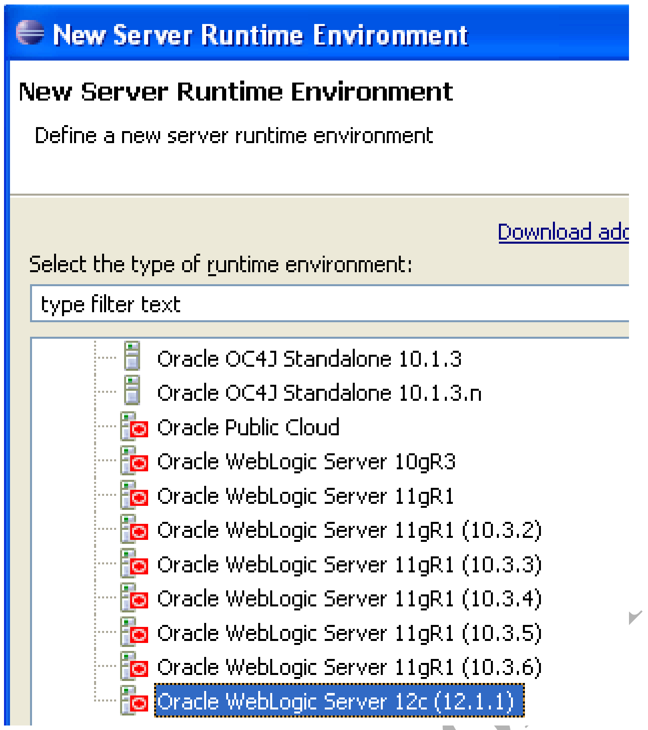


Add, remove, or edit server runtime environments.

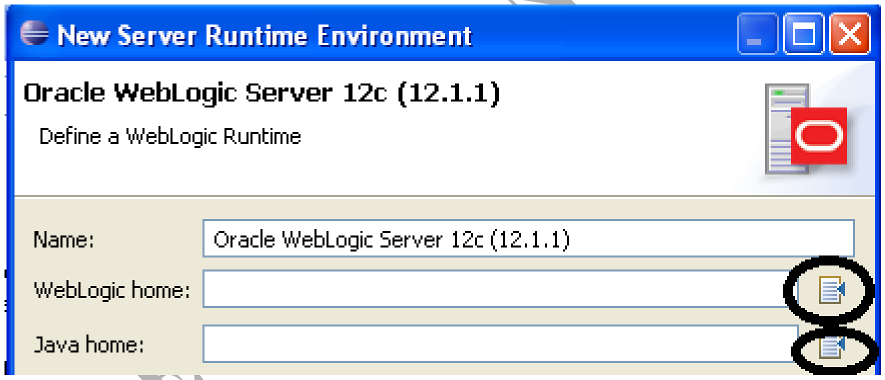
Server runtime environments:

__8. Click **Add...**

__9. Expand **Oracle** and select **Oracle Weblogic Server 12c(12.1.1)**


New Server Runtime Environment
New Server Runtime Environment
Define a new server runtime environment
Download adc
Select the type of runtime environment:
type filter text
Oracle OC4J Standalone 10.1.3
Oracle OC4J Standalone 10.1.3.n
Oracle Public Cloud
Oracle WebLogic Server 10gR3
Oracle WebLogic Server 11gR1
Oracle WebLogic Server 11gR1 (10.3.2)
Oracle WebLogic Server 11gR1 (10.3.3)
Oracle WebLogic Server 11gR1 (10.3.4)
Oracle WebLogic Server 11gR1 (10.3.5)
Oracle WebLogic Server 11gR1 (10.3.6)
Oracle WebLogic Server 12c (12.1.1)

__10. Click **Next**.


New Server Runtime Environment
Oracle WebLogic Server 12c (12.1.1)
Define a WebLogic Runtime

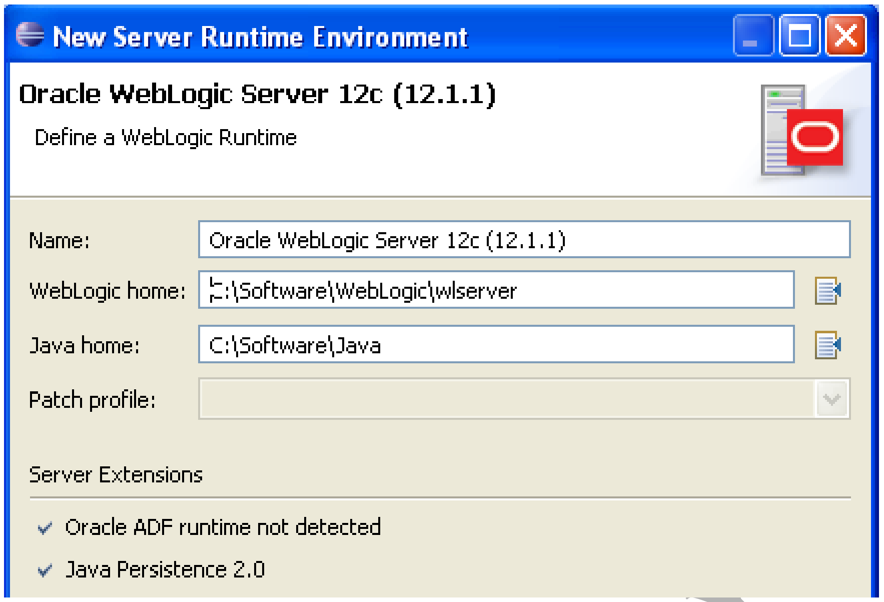
Name: Oracle WebLogic Server 12c (12.1.1)

WebLogic home:

Java home:

__11. For Weblogic home, enter **C:\Software\WebLogic\wlserver** or browse to find the location. Similarly enter **C:\Software\Java** for Java home.

You will see as shown below:



New Server Runtime Environment

Oracle WebLogic Server 12c (12.1.1)

Define a WebLogic Runtime

Name: Oracle WebLogic Server 12c (12.1.1)

WebLogic home: C:\Software\WebLogic\wlserver

Java home: C:\Software\Java

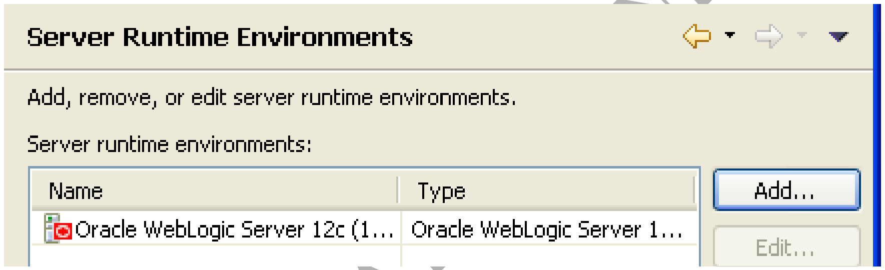
Patch profile:

Server Extensions

✓ Oracle ADF runtime not detected

✓ Java Persistence 2.0

__12. Click **Finish**.



Server Runtime Environments

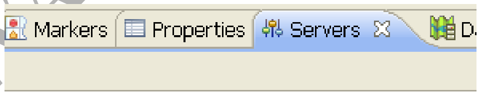
Add, remove, or edit server runtime environments.

Server runtime environments:

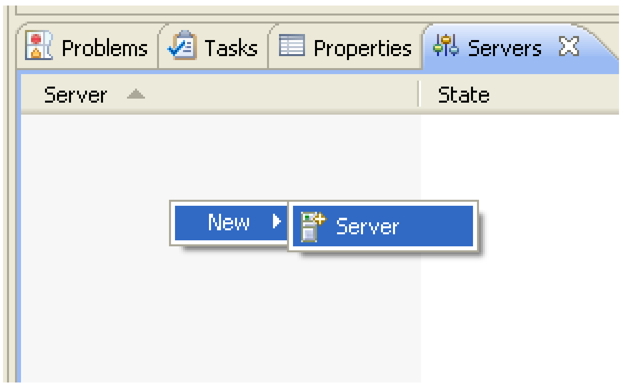
Name	Type
Oracle WebLogic Server 12c (1...	Oracle WebLogic Server 1...

__13. Click **OK**.

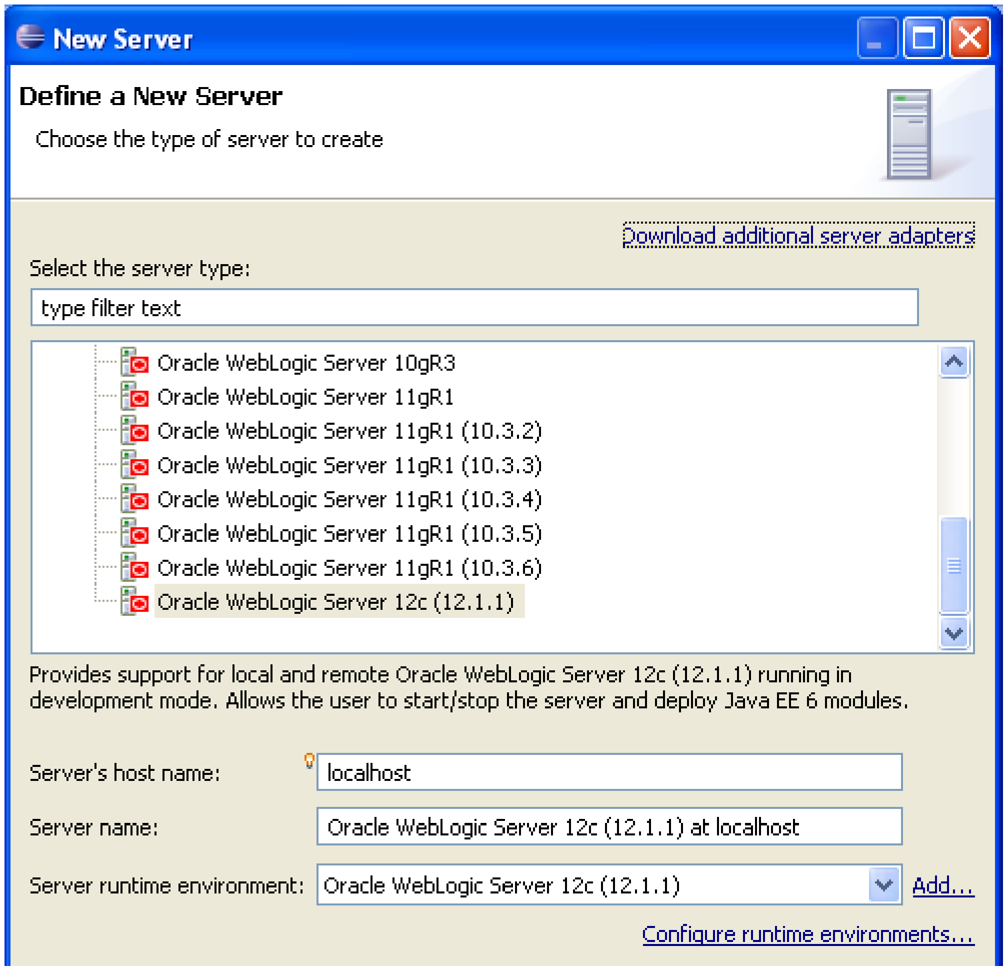
__14. We need to create domain and configure OEPE to point at the domain. Locate the *Server view*. To do this, click the **Servers** tab in the bottom area of OEPE.



__15. Right click anywhere in the area and select **New | Server**.



The *New Server* window will appear.



Define a New Server

Choose the type of server to create

Download additional server adapters

Select the server type:

type filter text

- Oracle WebLogic Server 10gR3
- Oracle WebLogic Server 11gR1
- Oracle WebLogic Server 11gR1 (10.3.2)
- Oracle WebLogic Server 11gR1 (10.3.3)
- Oracle WebLogic Server 11gR1 (10.3.4)
- Oracle WebLogic Server 11gR1 (10.3.5)
- Oracle WebLogic Server 11gR1 (10.3.6)
- Oracle WebLogic Server 12c (12.1.1)

Provides support for local and remote Oracle WebLogic Server 12c (12.1.1) running in development mode. Allows the user to start/stop the server and deploy Java EE 6 modules.

Server's host name: localhost

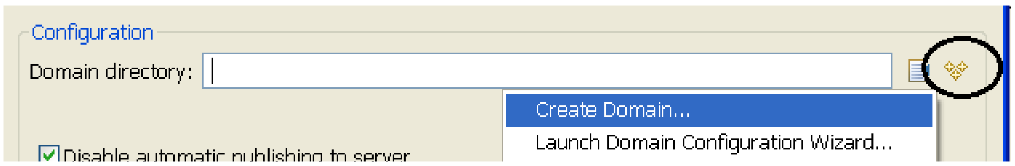
Server name: Oracle WebLogic Server 12c (12.1.1) at localhost

Server runtime environment: Oracle WebLogic Server 12c (12.1.1) Add...

Configure runtime environments...

__16. Select **Oracle WebLogic Server 12c**.

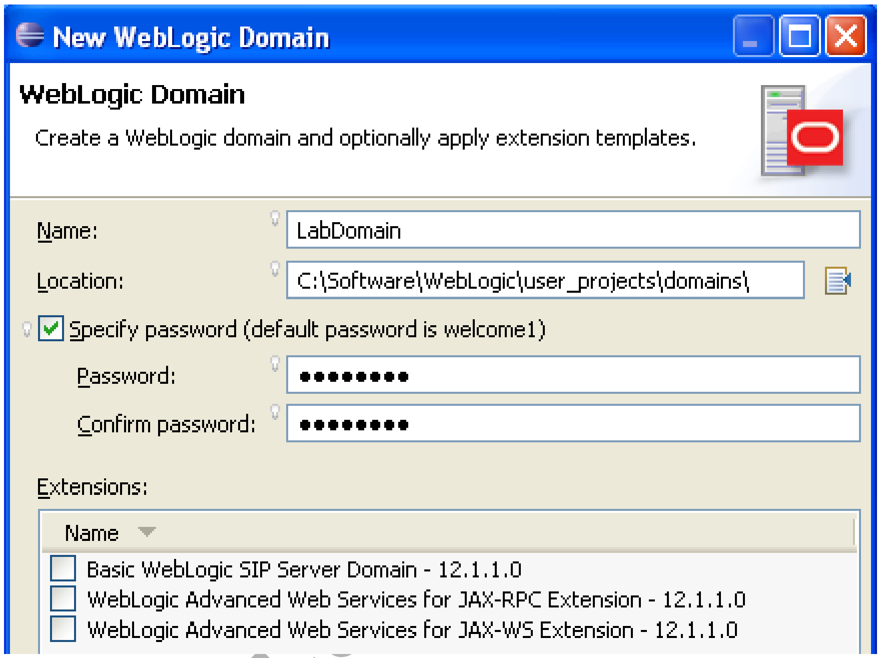
__17. Click **Next**.


Configuration
Domain directory:
Create Domain...
Launch Domain Configuration Wizard...
Disable automatic publishing to server

__18. Next to **Domain directory**, click the **create** icon and then select **Create Domain**.

__19. Enter **LabDomain** as the name.

__20. Check specify password and enter **w1sadmin** as password **where the password contains number one not L.**


New WebLogic Domain

WebLogic Domain

Create a WebLogic domain and optionally apply extension templates.

Name: LabDomain

Location: C:\Software\WebLogic\user_projects\domains\

[✓] Specify password (default password is welcome1)

Password: ●●●●●●●●●

Confirm password: ●●●●●●●●●

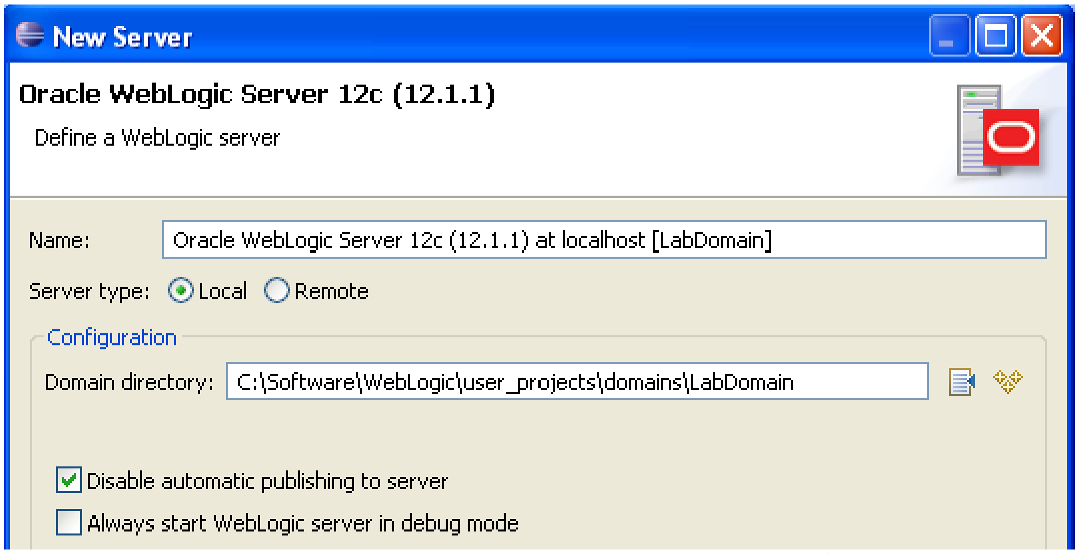
Extensions:

Name
☐ Basic WebLogic SIP Server Domain - 12.1.1.0
☐ WebLogic Advanced Web Services for JAX-RPC Extension - 12.1.1.0
☐ WebLogic Advanced Web Services for JAX-WS Extension - 12.1.1.0

__21. Don't select any extensions, because we will not be using any of the advanced features like WS-Addressing.

__22. Click **Finish**.

__23. After some seconds the below screen will open, make sure **Disable Automatic Publishing to server** is checked.


New Server

Oracle WebLogic Server 12c (12.1.1)

Define a WebLogic server

Name: Oracle WebLogic Server 12c (12.1.1) at localhost [LabDomain]

Server type: [✓ Local] Remote

Configuration

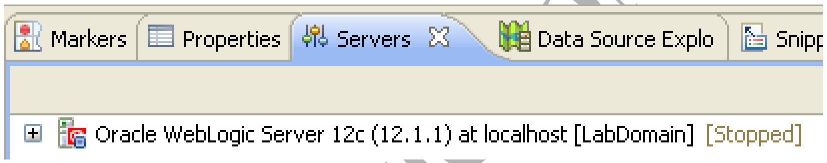
Domain directory: C:\Software\WebLogic\user_projects\domains\LabDomain

[✓] Disable automatic publishing to server

[ ] Always start WebLogic server in debug mode

__24. Click **Finish**.

The server connection has been defined, and the server should now be listed in the *Servers* view.

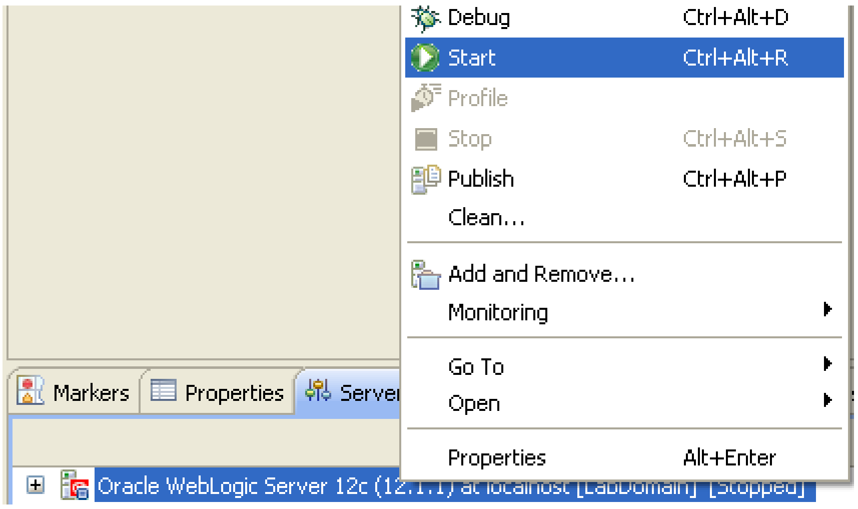


Oracle WebLogic Server 12c (12.1.1) at localhost [LabDomain] [Stopped]

Part 3 - Working with the Server

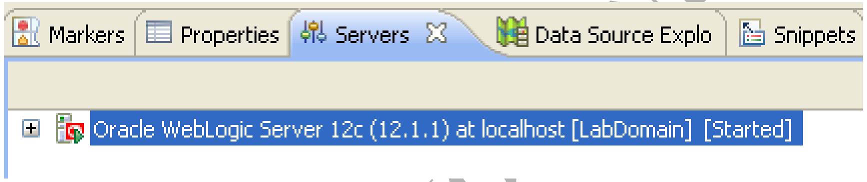
Throughout this class, we will be using this view to perform 3 basic operations: server *stop*, *start* and *publish*. We will discuss publishing in a later lab, so let us now focus on starting and stopping the server.

__1. Let us start things off by starting the server. In the *Servers* view, right click on the server and select **Start**.

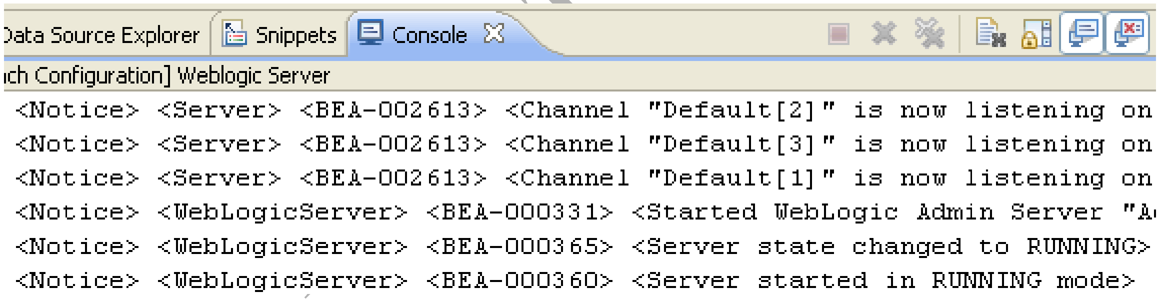


The *Console* view should automatically open. This will allow us to “see” OEPE booting. Eventually, the server will start and the *Servers* view will be brought to the focus again.

After finish publishing the server *State* should be **Started**.


Markers
Properties
Servers
Data Source Explo
Snippets
Oracle WebLogic Server 12c (12.1.1) at localhost [LabDomain] [Started]

__2. Examine the *Console* window by clicking the *Console* tab.

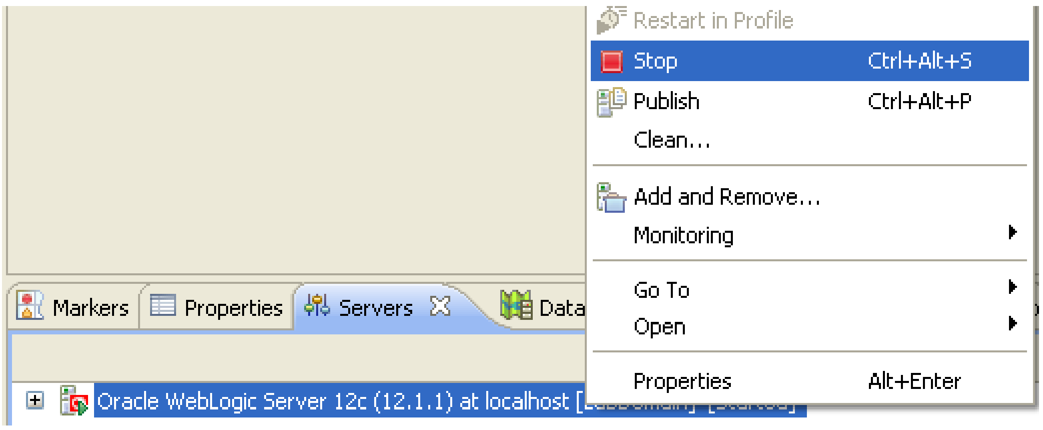


```
<Notice> <Server> <BEA-002613> <Channel "Default[2]" is now listening on
<Notice> <Server> <BEA-002613> <Channel "Default[3]" is now listening on
<Notice> <Server> <BEA-002613> <Channel "Default[1]" is now listening on
<Notice> <WebLogicServer> <BEA-000331> <Started WebLogic Admin Server "A
<Notice> <WebLogicServer> <BEA-000365> <Server state changed to RUNNING>
<Notice> <WebLogicServer> <BEA-000360> <Server started in RUNNING mode>
```

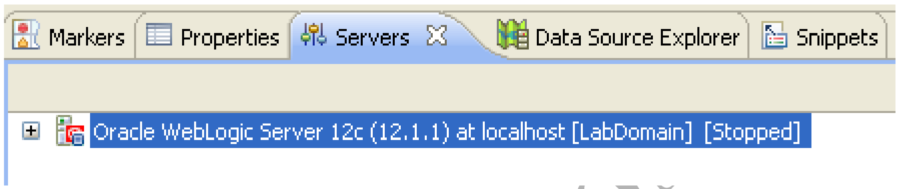
In later lab steps, when we ask you to refer to the *server console*, this is the view we will be referring to.

__3. Let us try stopping the server. Go back to the *Servers* view.

__4. Right click on the server and select **Stop**.



As with starting the server, the console view will open briefly. Then the *Server* view will appear, listing the status as *Stopped*.


Markers
Properties
Servers
Data Source Explorer
Snippets
Oracle WebLogic Server 12c (12.1.1) at localhost [LabDomain] [Stopped]

You have stopped the server.

Part 4 - Review

In this lab, you configured the development environment.

You started things off by configuring environment followed by WebLogic Server domain creation and server setup, which will be used to host our web service applications.

You saw how to start and stop a WebLogic Server instance.

Lab 2 - Develop a Simple RESTful Service

Over the next few labs, our goal will be to learn the fundamentals of the JAX-RS API. We will not get into implementing the services for Acme Inc. just yet.

In this lab, we will develop a very simple RESTful web service. This will let us focus on the development process using the OEPE tool and WebLogic server. We will also learn about the basic JAX-RS annotations.

The main goal of this lab is to understand how to create a web service project and configure it properly for JAX-RS development.

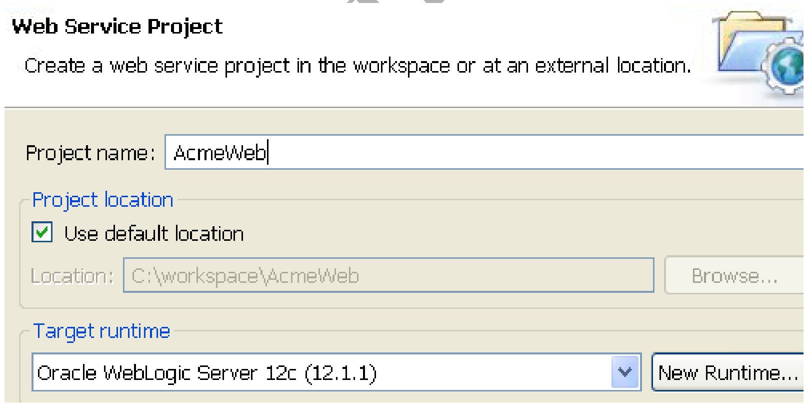
Part 1 - Create the Web Service Project

In OEPE, we can develop a JAX-RS web service inside of a dynamic web project or a web service project. The latter is just an extension of a dynamic web project with a few extra JAR files added to the compiler's build path. These extra JAR files include the Jersey 1.9 JAR files that implement the JAX-RS API. Hence, using the web service project will save us a bit of time. If you have a legacy dynamic web project where you will like to create a JAX-RS service, all you have to do is add the Jersey JAR files to the build path. We will look into that shortly.

__1. Launch OEPE if it is not already running.

__2. Switch to the **Java EE** perspective.

__3. From the menubar, select **File > New > Web Service Project**.



Web Service Project

Create a web service project in the workspace or at an external location.

Project name: AcmeWeb

Project location

☑ Use default location

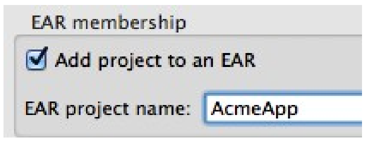
Location: C:\workspace\AcmeWeb

Target runtime

Oracle WebLogic Server 12c (12.1.1)

__4. Enter **AcmeWeb** as the project name.

__5. Ensure that **Oracle WebLogic Server 12c** as the targeted runtime.



EAR membership

☑ Add project to an EAR

EAR project name: AcmeApp

__6. Add the project to an enterprise application archive project called **AcmeApp**

Tip

You must add the web service project to a EAR. Otherwise, it appears, the JAX-RS services don't work.

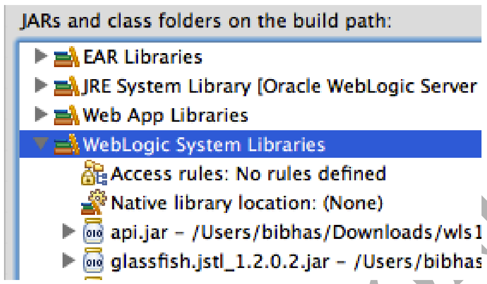
__7. Click **Finish**.

We will briefly review the project.

__8. Right click the newly created AcmeWeb project and select **Properties**.

__9. Select the **Java Build Path** properties.

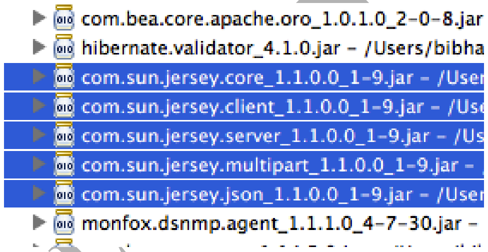
__10. Click the **Libraries** tab.



JARs and class folders on the build path:

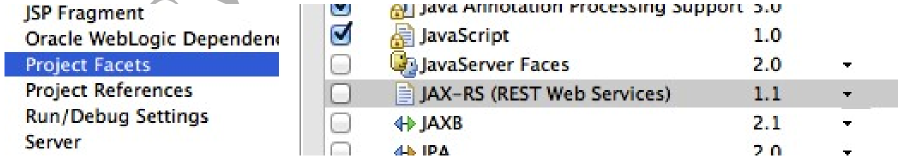
- EAR Libraries
- JRE System Library [Oracle WebLogic Server
- Web App Libraries
- WebLogic System Libraries
  - Access rules: No rules defined
  - Native library location: (None)
  - api.jar - /Users/bibhas/Downloads/wls1
  - glassfish.jstl_1.2.0.2.jar - /Users/bibhas

__11. Expand **WebLogic Systems Library**.



- com.bea.core.apache.oro_1.0.1.0_2-0-8.jar
- hibernate.validator_4.1.0.jar - /Users/bibha
- com.sun.jersey.core_1.1.0.0_1-9.jar - /User
- com.sun.jersey.client_1.1.0.0_1-9.jar - /Use
- com.sun.jersey.server_1.1.0.0_1-9.jar - /Us
- com.sun.jersey.multipart_1.1.0.0_1-9.jar -
- com.sun.jersey.json_1.1.0.0_1-9.jar - /User
- monfox.dsnmp.agent_1.1.1.0_4-7-30.jar -

__12. Scroll down and locate the **Jersey JAR** files. If you use a plain dynamic web project, you will need to manually add these JAR files to the build path.



JSP Fragment
Oracle WebLogic Dependen
Project Facets
Project References
Run/Debug Settings
Server

✓	Java Annotation Processing Support	5.0
✓	JavaScript	1.0
☐	JavaServer Faces	2.0
☐	JAX-RS (REST Web Services)	1.1
☐	JAXB	2.1
☐	JPA	2.0

__13. Now, select the **Project Facets** property.

__14. Note that the JAX-RS facet is not selected. This facet plays no role in developing JAX-RS services for WebLogic.

__15. Click **Cancel** to close the dialog.

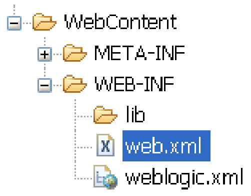
Part 2 - Register the REST Handler Servlet

A REST servlet handles all incoming request for RESTful services. In our case, this servlet is available from the Jersey implementation. All we have to do is register it in web.xml. To save time, this file is given to you. You will simply import it.

__1. Open Windows file explorer.

__2. Go to **C:\LabFiles\**

__3. Copy **web.xml**



__4. In Eclipse OEPE, paste the file inside the **WebContent > WEB-INF** folder of **AcmeWeb**.

__5. Open **web.xml** and study how the servlet is registered. Specifically, note the servlet mapping:

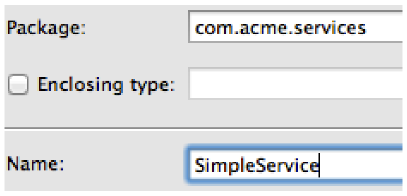
```
<servlet-mapping>
    <servlet-name>RestServlet</servlet-name>
    <url-pattern>/svc/*</url-pattern>
</servlet-mapping>
```

This means, the URL for every REST request will start with http://host:port/AcmeWeb/svc/. You can choose any other path for the REST servlet. But, we will stick to the short and sweet "svc".

Part 3 - Create the Resource Class

The Java class that implements a RESTful service is called a resource. We will now develop a Java class for a simple service.

__1. Right click **AcmeWeb** project and select **New > Class**.


Package: com.acme.services
Enclosing type:
Name: SimpleService

__2. Enter **com.acme.services** as the package name and **SimpleService** as the class name.

__3. Click **Finish**.

__4. Add a member variable that will help us do logging.

```
Logger logger = Logger.getLogger("SimpleService");
```

We will now add the testGET() method. It will not have any business logic. Later, we will map this method to a GET request.

__5. Add the method as follows.

```
public String testGET() {
          logger.info("Got a GET request");

          return "OK";
}
```

__6. Organize imports (Control+Shift+O) and select **java.util.logging.Logger**.

__7. Save changes.

Part 4 - Configure the Resource

We will now configure the URI and HTTP method for the service resource and its methods. We will use the "/simple" root URI for the service.

__1. Define the URI of the root resource, by adding the @Path annotation above the class. This is shown in bold face below.

```
@Path("/simple")
public class SimpleService {
```

The testGET() method does not need any path extension. All we have to do is set GET as the HTTP method.

We will also set "text/plain" as the content MIME type of the reply. That is good enough for this method. For XML data type, the MIME will be "text/xml".

__2. Set the HTTP method and reply MIME type for the testGET() method sub-resource as follows.

```
@GET
@Produces("text/plain")
public String testGET() {
```

__3. Organize imports. Select **javax.ws.rs.Produces**.

__4. Save changes.

Part 5 - Unit Test

We will now exercise the web service from a browser.

__1. Start the server if it is not running.

__2. Right click the server and select **Add and Remove**.

__3. Add the **AcmeApp** project to the server.

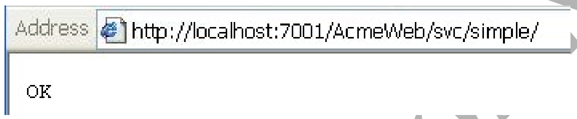
__4. Click **Finish**.

__5. Make sure the server is synchronized.

__6. Open a web browser and enter the URL:

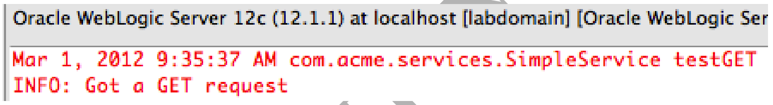
```
http://localhost:7001/AcmeWeb/svc/simple/
```

__7. You should see OK in the browser.



```
http://localhost:7001/AcmeWeb/svc/simple/

OK
```

__8. The Console view in Eclipse will show the log output.


Oracle WebLogic Server 12c (12.1.1) at localhost [labdomain] [Oracle WebLogic Ser

```
Mar 1, 2012 9:35:37 AM com.acme.services.SimpleService testGET
INFO: Got a GET request
```

This proves that our project has been setup correctly and JAX-RS is working fine.

Part 6 - Use Path Extension for Sub-resource

A sub-resource – Java method – can be mapped to an URI extension path. Any HTTP request for that path will be handled by that method. We will now create a method that will respond to the /simple/**mypath** URI.

__1. First, add this method.

```
public String testGETWithPath() {
          logger.info("Got a GET request with path extension.");

          return "OK";
}
```

__2. Annotate the method as follows.

```
@GET
@Produces("text/plain")
@Path("/mypath")
```

```
public String testGETWithPath() {
```

__3. Save changes.

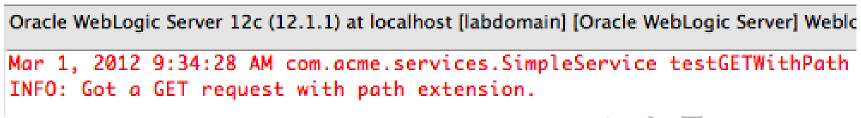
Part 7 - Unit Test

__1. Right click the server and select **Publish**.

__2. In a web browser, enter the URL:

```
http://localhost:7001/AcmeWeb/svc/simple/mypath
```

__3. Make sure that the Console shows the log output.



```
Oracle WebLogic Server 12c (12.1.1) at localhost [labdomain] [Oracle WebLogic Server] Weblc
Mar 1, 2012 9:34:28 AM com.acme.services.SimpleService testGETWithPath
INFO: Got a GET request with path extension.
```

__4. Open a new browser and re-test the testGET method for regression using the URL:

```
http://localhost:7001/AcmeWeb/svc/simple/
```

__5. Close all open files.

__6. Close all browsers.

Part 8 - Review

In this lab, we created and configured a web service project. We created a very simple JAX-RS web service. At this point, you should know how to configure the URI of the root resource – the Java class - using the @Path annotation. Also, we configured the HTTP method of a sub-resource – a method – using the @GET annotation.

Lab 3 - Extracting Information from a HTTP Request

RESTful services are executed by sending HTTP requests. The request contains input data in various areas:

1. In the URI path. For example: /orders/**1051**.
2. As URL parameters. For example: /orders?**status=P**
3. Input data from a form submission.
4. Less commonly, from HTTP header and cookie.

We will now learn how to extract data from common locations.

Part 1 - Get Root Resource Path Parameters

Input data can be added to the path of the root resource as well as the path of a method (sub-resource). In REST, data is added to the URI to form an unique identifier.

First, we will add an input parameter in the root path of the SimpleService resource. The root URI will now look like /simple/**somedata**. The root URI will continue to execute the testGET() method since this method defines no path extension. To execute the testGETWithPath(), the URI will need to be /simple/**somedata**/mypath.

__1. Open **SimpleService.java**

__2. Change the @Path annotation for the class and add a parameter there.

```
@Path("/simple/{myRootPathData}")
public class SimpleService {
```

Here, {myRootPathData} is a placeholder for a parameter. You can use anything as a name of the parameter. The name will play a role to obtain the value of the parameter.

The best place to capture input from a root path is a member variable of the resource. We will do that now.

__3. Add a member variable as follows.

```
String rootPathData;
```

__4. Annotate the member variable to save the value of the myRootPathData parameter.

```
@PathParam("myRootPathData")
String rootPathData;
```

That's it. Now, JAX-RS will extract the value of the parameter from the URI and set it to the member variable right after the resource object is created. By default, a POJO resource instance is created for every HTTP request. Hence, every request can have a different parameter value in the path.

__5. Change the log statement of the testGET() method as follows.

```
logger.info("Got a GET request with root path data: " + rootPathData);
```

__6. Make a similar change to the testGETWithPath() method.

```
logger.info(
    "Got a GET request with path extension with root path data: " +
    rootPathData);
```

__7. Organize imports.

__8. Save changes.

Part 2 - Unit Test

__1. Publish the server.

__2. First, test the testGET() method by entering the URL:

```
http://localhost:7001/AcmeWeb/svc/simple/somedata/
```

__3. The log output will like this:

```
INFO: Got a GET request with root path data: somedata
```

__4. Now, test the testGETWithPath() method using the URL:

```
http://localhost:7001/AcmeWeb/svc/simple/somedata/mypath
```

__5. The log output will be:

```
INFO: Got a GET request with path extension with root path data:
somedata
```

Part 3 - Get Sub-resource Path Parameters

Methods can also let us add parameters to its path. For example to get the billing address of an order #1051, we can use the URI: /orders/**1051**/address/**billing**. Never lose sight of the fact that a URI uniquely identifies an entity or a collection of entities. In this example, we are specifically pointing to the billing address used with an order. Here, the orderId as well as the type of address requested can be added as parameters to the path of the sub-resource. Let's try out this example.

__1. Add a basic method as follows.

```
public String getAddress(
            int oId,
            String type) {
            logger.info("Order Id: " + oId);
            logger.info("Address type: " + type);

            return "OK";
}
```

__2. Annotate method with path and method.

```
@GET
@Produces("text/plain")
@Path("/{orderId}/address/{addressType}")
public String getAddress(
```

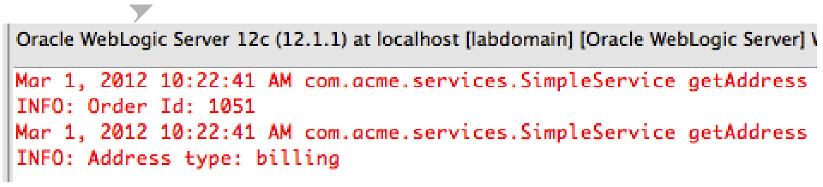
__3. Now, extract the path parameters and save them in the two input argument variables of the method.

```
public String getAddress(
            @PathParam("orderId")
            int oId,
            @PathParam("addressType")
            String type) {
```

__4. Save changes.

__5. Test the method by entering the URL:

```
http://localhost:7001/AcmeWeb/svc/simple/somedata/1051/address/billing
```


Oracle WebLogic Server 12c (12.1.1) at localhost [labdomain] [Oracle WebLogic Server]

```
Mar 1, 2012 10:22:41 AM com.acme.services.SimpleService getAddress
INFO: Order Id: 1051
Mar 1, 2012 10:22:41 AM com.acme.services.SimpleService getAddress
INFO: Address type: billing
```

Part 4 - Extract Query Parameters

Query parameters are available from the requested URI as:

```
GET /path?param1=value&param2=value2 HTTP/1.1
```

You can capture them either using resource class member variable or method argument variable. The latter is generally recommended for better code readability. We will try that out now.

__1. Add the giveRaise() method as shown below.

```
@GET
@Produces("text/plain")
@Path("raise")
public String giveRaise(
    @QueryParam("name")
    String employeeName,
    @QueryParam("amount")
    double amount
    ) {
    logger.info("Giving raise to " + employeeName + " by " + amount);

    return "OK";
}
```

__2. Organize imports.

__3. Save changes.

__4. Test the change by entering the URL:

```
http://localhost:7001/AcmeWeb/svc/simple/somedata/raise?name=Daffy&amount=10000
```

__5. Make sure that the Console shows:

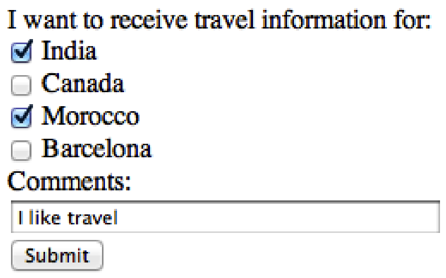
```
INFO: Giving raise to Daffy by 10000.0
```

Note, how JAX-RS converted the amount URL parameter from text to double.

Part 5 - Extracting Form Post Data

Form data is available in the HTTP request body. The data is encoded using MIME type application/x-www-form-urlencoded. Usually, such data is submitted using the POST method.

Note: The Java EE Servlet specification does not distinguish between URL query parameter and POST data in HTTP request body. They are both accessed using request.getParameter() method. JAX-RS, however, makes a distinction. Query parameter can be accessed using @QueryParam. Form POST data has to be accessed using @FormParam.



We will now build a service method that will accept input from the form above.

__1. Add this method.

```
@POST
@Produces("text/plain")
@Path("feedback")
public String submitFeed(
        @FormParam("interest")
        List<String> interestList,
        @FormParam("comments")
        String comments
    ) {
    for (String interest : interestList) {
        logger.info("Interest: " + interest);
    }
    logger.info("Comments: " + comments);

    return "OK";
}
```

Note that the HTTP method is set to POST. Also, we expect multiple values for the "interest" parameter. That is why we have set the data type of the variable to java.util.List<String>.

__2. Organize imports. Select **java.util.List**.

__3. Save changes.

Part 6 - Unit Test

__1. We will import the form HTML page. Copy **C:\LabFiles\post.html** and paste it inside the **WebContent** folder of AcmeWeb project.

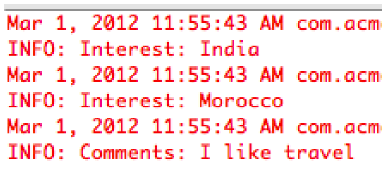
__2. In the *Servers* view, right click the server and select **Publish**.

__3. Open a browser and enter the URL:

```
http://localhost:7001/AcmeWeb/post.html
```

__4. Fill out the form and submit it.

__5. Make sure that the console log shows the input values correctly.



```
Mar 1, 2012 11:55:43 AM com.acm
INFO: Interest: India
Mar 1, 2012 11:55:43 AM com.acm
INFO: Interest: Morocco
Mar 1, 2012 11:55:43 AM com.acm
INFO: Comments: I like travel
```

__6. Close all open files.

__7. Close all open browsers.

Part 7 - Review

In this lab, we learned how to gather input data from a couple of common sources:

1. URI path of the root resource
2. URI path of the method sub-resource.
3. URL query parameter.
4. Form POST data.