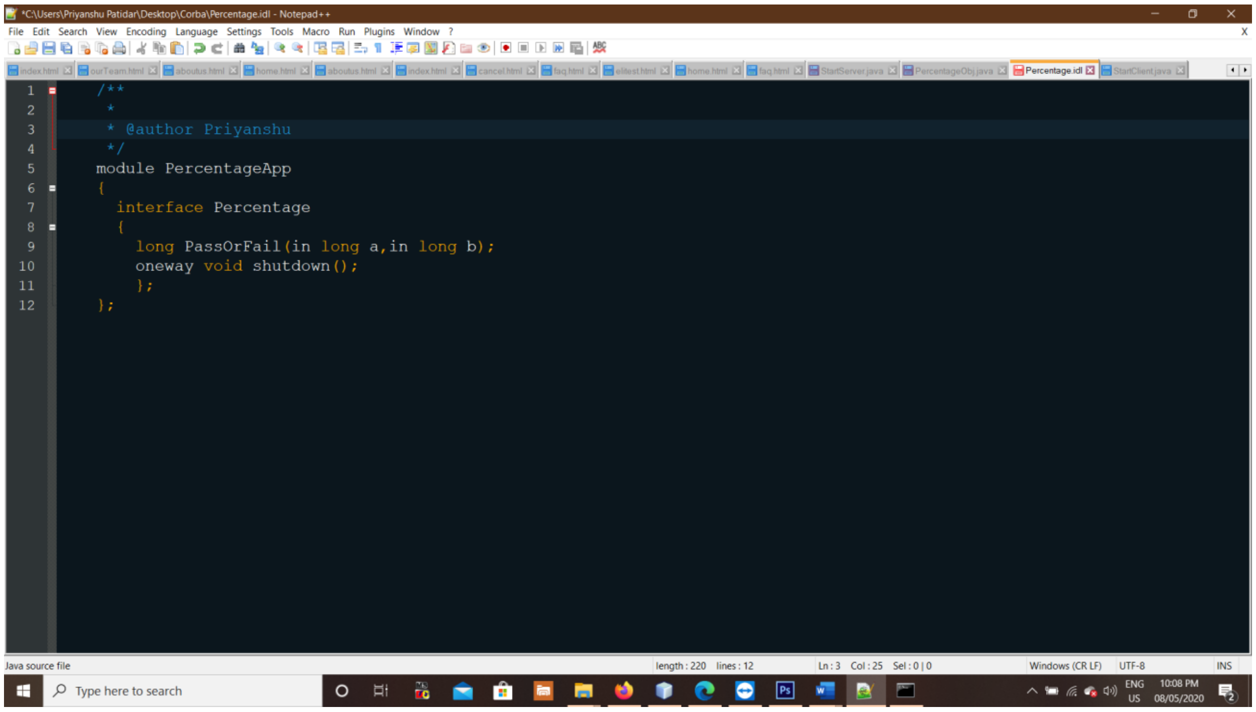
Q1. Describe the various steps involved in a java programming with CORBA to implement the following: A client sent a name and five subject marks to the server, server returns the message fail or pass status to the client. If all the marks are greater than 50 than “Pass” else “Fail” For example, the client passes “Ram 56 67 78 89 90”, the server replies as “Ram, you pass your examinations.

Answer:-

1. Server Side

1. Let’s create a new **Java Project** using Eclipse (or NetBeans or other editor you prefer), and call it : **CorbaPercentageServer**-> Click **Finish** once done
2. Under the project **CorbaPercentageServer**, Create a new file called **Percentage.idl**. Copy the code below into the new file.

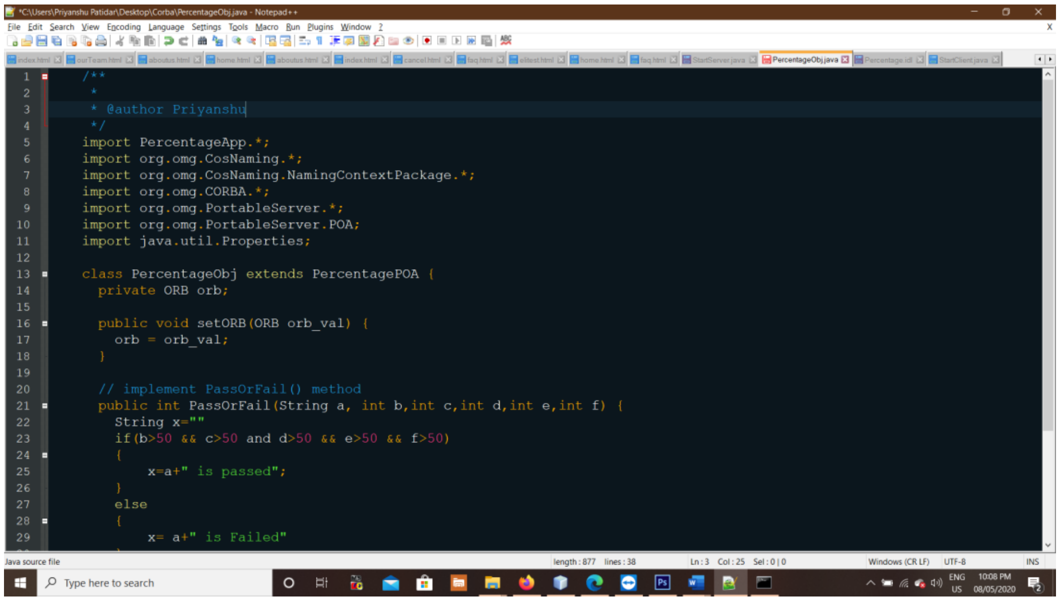

*C:\Users\Priyanshu Patidar\Desktop\Corba\Percentage.idl - Notepad++

```
/**
 *
 * @author Priyanshu
 */
module PercentageApp
{
  interface Percentage
  {
    long PassOrFail(in long a,in long b);
    oneway void shutdown();
    };
};
```

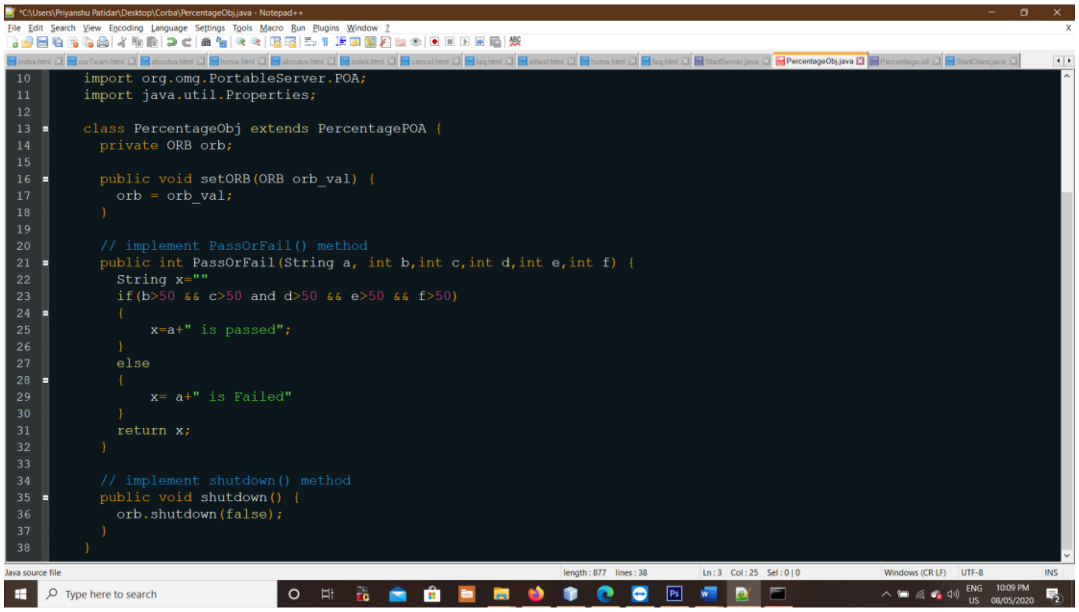
3. Open your CMD console, Change directory (cd) to **src** folder within the project location. The location of your project can be known through clicking: Select the project CorbaPercentageServer, click : **File** -> **Properties**
4. 4 Compile the idl file using the idlj command as below:

```
idlj -fall Percentage.idl
```

5. Right click on the **src** folder and select **Refresh**
6. Under the project **CorbaPercentageServer**, Create a new **Class** called **PercentageObj**. Copy the code below into the new class.

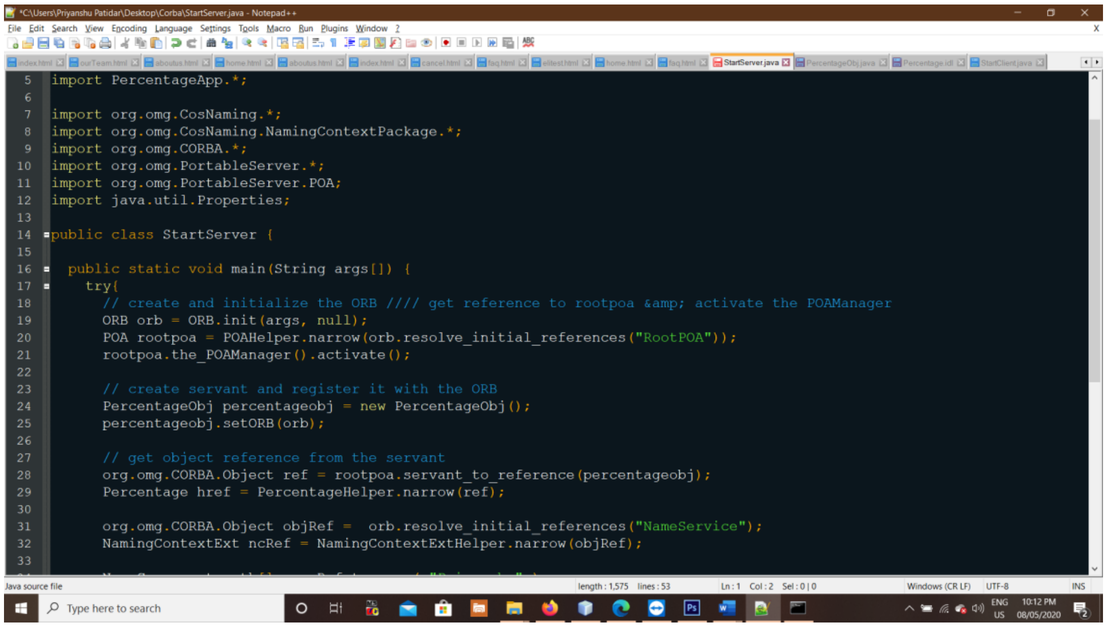

*C:\Users\Priyanshu Patidar\Desktop\Corba\PercentageObj.java - Notepad++

```
/**
 *
 * @author Priyanshu
 */
import PercentageApp.*;
import org.omg.CosNaming.*;
import org.omg.CosNaming.NamingContextPackage.*;
import org.omg.CORBA.*;
import org.omg.PortableServer.*;
import org.omg.PortableServer.POA;
import java.util.Properties;

class PercentageObj extends PercentagePOA {
  private ORB orb;

  public void setORB(ORB orb_val) {
    orb = orb_val;
  }

  // implement PassOrFail() method
  public int PassOrFail(String a, int b,int c,int d,int e,int f) {
    String x=""
    if(b>50 && c>50 and d>50 && e>50 && f>50)
    {
        x=a+" is passed";
    }
    else
    {
        x= a+" is Failed"
```

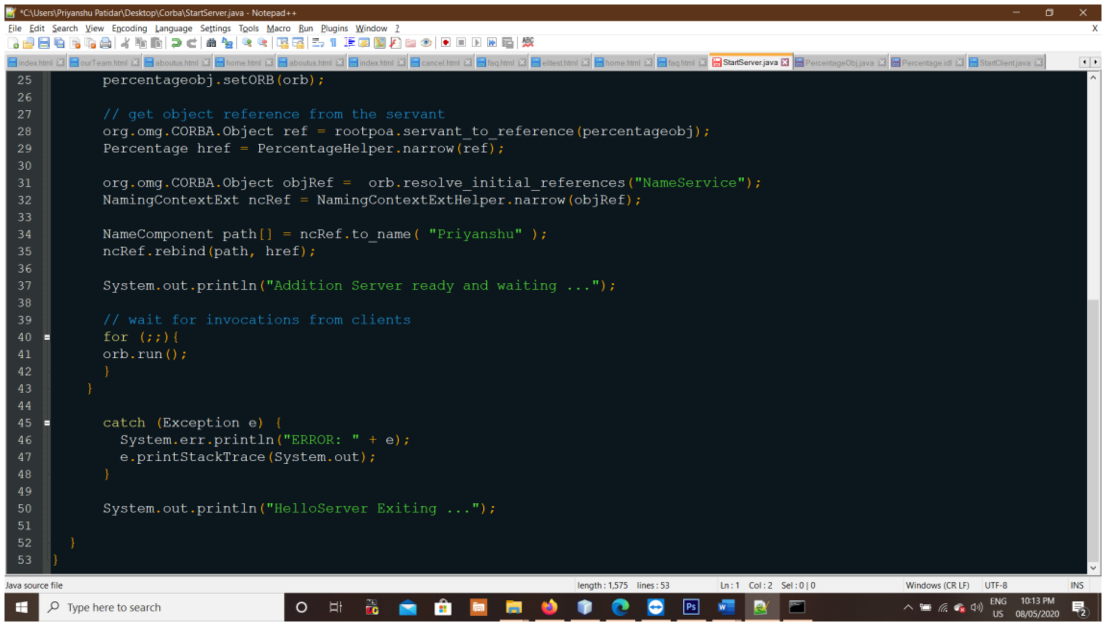

*C:\Users\Priyanshu Patidar\Desktop\Corba\PercentageObj.java - Notepad++

```
import org.omg.PortableServer.POA;
import java.util.Properties;

class PercentageObj extends PercentagePOA {
  private ORB orb;

  public void setORB(ORB orb_val) {
    orb = orb_val;
  }

  // implement PassOrFail() method
  public int PassOrFail(String a, int b,int c,int d,int e,int f) {
    String x=""
    if(b>50 && c>50 and d>50 && e>50 && f>50)
    {
        x=a+" is passed";
    }
    else
    {
        x= a+" is Failed"
    }
    return x;
  }

  // implement shutdown() method
  public void shutdown() {
    orb.shutdown(false);
  }
}
```

7. Under the project **CorbaPercentageServer**, Create a new **Class** called **StartServer**. Copy the code below into the new class.



```java
import PercentageApp.*;

import org.omg.CosNaming.*;
import org.omg.CosNaming.NamingContextPackage.*;
import org.omg.CORBA.*;
import org.omg.PortableServer.*;
import org.omg.PortableServer.POA;
import java.util.Properties;

public class StartServer {

  public static void main(String args[]) {
    try{
      // create and initialize the ORB //// get reference to rootpoa &amp; activate the POAManager
      ORB orb = ORB.init(args, null);
      POA rootpoa = POAHelper.narrow(orb.resolve_initial_references("RootPOA"));
      rootpoa.the_POAManager().activate();

      // create servant and register it with the ORB
      PercentageObj percentageobj = new PercentageObj();
      percentageobj.setORB(orb);

      // get object reference from the servant
      org.omg.CORBA.Object ref = rootpoa.servant_to_reference(percentageobj);
      Percentage href = PercentageHelper.narrow(ref);

      org.omg.CORBA.Object objRef =  orb.resolve_initial_references("NameService");
      NamingContextExt ncRef = NamingContextExtHelper.narrow(objRef);

```



```java
      percentageobj.setORB(orb);

      // get object reference from the servant
      org.omg.CORBA.Object ref = rootpoa.servant_to_reference(percentageobj);
      Percentage href = PercentageHelper.narrow(ref);

      org.omg.CORBA.Object objRef =  orb.resolve_initial_references("NameService");
      NamingContextExt ncRef = NamingContextExtHelper.narrow(objRef);

      NameComponent path[] = ncRef.to_name( "Priyanshu" );
      ncRef.rebind(path, href);

      System.out.println("Addition Server ready and waiting ...");

      // wait for invocations from clients
      for (;;){
      orb.run();
      }
    }

      catch (Exception e) {
        System.err.println("ERROR: " + e);
        e.printStackTrace(System.out);
      }

      System.out.println("HelloServer Exiting ...");

  }
}
```

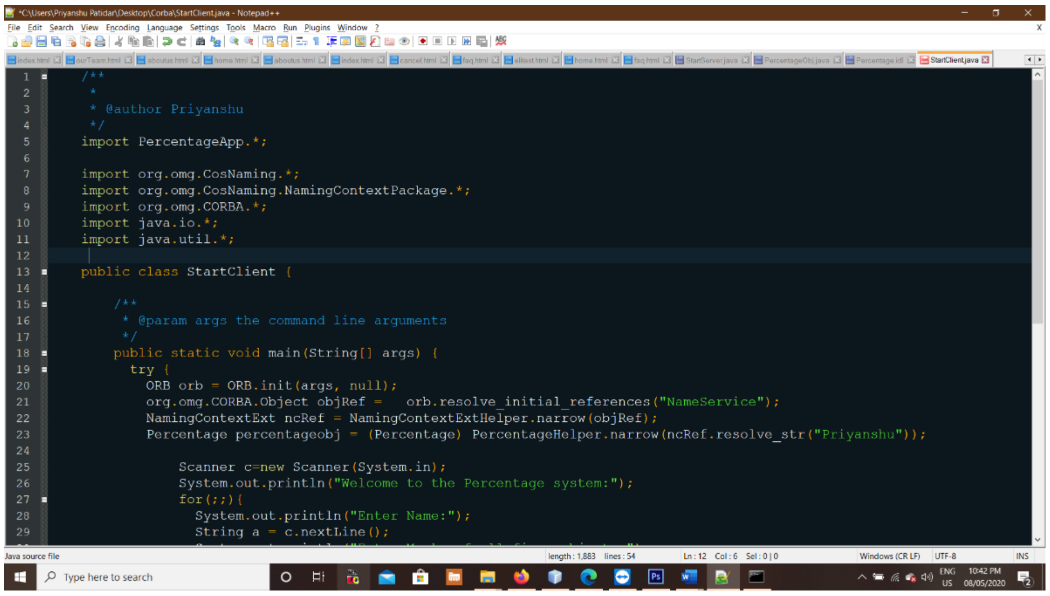
Name-Priyanshu Patidar
Enrollment Number-17100BTCSE01253

8. Open a new windows of **CMD console,** type in the following command to start the **ORB**
   start orbd -ORBInitialPort 1050
9. Under Eclipse now, click on the project **CorbaPercentageServer,** then Click **Run -> Run Configuration**
10. Make sure you select the **StartServer** for the **CorbaPercentageServer.**
11. Click on Arguments and type in the following arguments inside the **Program arguments.** Click **Apply** once done:
    -ORBInitialPort 1050 -ORBInitialHost localhost

12. That's all for the server, it should be running.

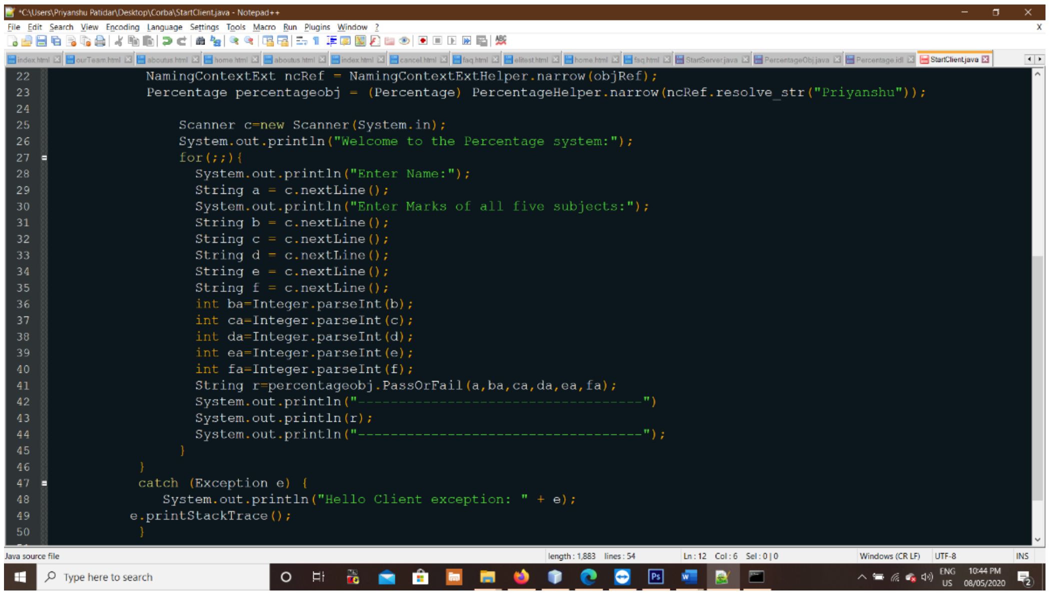
Client Side.

1. Let's create a new **Java Project** using Eclipse and call it : **CorbaPercentageClient**-> Click **Finish** once done
2. Right click on PercentageApp Package under CorbaPercentageServer and click Copy
3. **Right click on** src **Package under** CorbaPercentageClient **and click** Paste
4. Under the project CorbaPercentageClient, Create a new Class called StartClient. Copy the code below into the new class.


*C:\Users\Priyanshu Patidar\Desktop\Corba\StartClient.java - Notepad++

```java
/**
 *
 * @author Priyanshu
 */
import PercentageApp.*;

import org.omg.CosNaming.*;
import org.omg.CosNaming.NamingContextPackage.*;
import org.omg.CORBA.*;
import java.io.*;
import java.util.*;

public class StartClient {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
      try {
        ORB orb = ORB.init(args, null);
        org.omg.CORBA.Object objRef =   orb.resolve_initial_references("NameService");
        NamingContextExt ncRef = NamingContextExtHelper.narrow(objRef);
        Percentage percentageobj = (Percentage) PercentageHelper.narrow(ncRef.resolve_str("Priyanshu"));

            Scanner c=new Scanner(System.in);
            System.out.println("Welcome to the Percentage system:");
            for(;;){
              System.out.println("Enter Name:");
              String a = c.nextLine();
```

Java source file length : 1,883 lines : 54 Ln : 12 Col : 6 Sel : 0 | 0 Windows (CR LF) UTF-8 INS



```
        NamingContextExt ncRef = NamingContextExtHelper.narrow(objRef);
        Percentage percentageobj = (Percentage) PercentageHelper.narrow(ncRef.resolve_str("Priyanshu"));

            Scanner c=new Scanner(System.in);
            System.out.println("Welcome to the Percentage system:");
            for(;;){
              System.out.println("Enter Name:");
              String a = c.nextLine();
              System.out.println("Enter Marks of all five subjects:");
              String b = c.nextLine();
              String c = c.nextLine();
              String d = c.nextLine();
              String e = c.nextLine();
              String f = c.nextLine();
              int ba=Integer.parseInt(b);
              int ca=Integer.parseInt(c);
              int da=Integer.parseInt(d);
              int ea=Integer.parseInt(e);
              int fa=Integer.parseInt(f);
              String r=percentageobj.PassOrFail(a,ba,ca,da,ea,fa);
              System.out.println("-----------------------------------")
              System.out.println(r);
              System.out.println("-----------------------------------");
            }
        }
       catch (Exception e) {
          System.out.println("Hello Client exception: " + e);
      e.printStackTrace();
       }
```

5. Click on Run -> Run Configuration, Make sure you select the StartClient for the CorbaAdditionClient.
6. Click on Arguments, then under the Program arguments, type in the following :
   -ORBInitialPort 1050 -ORBInitialHost localhost

7. Click Apply, then Run. That's all for the client, it should be running.

Q2. Give any two weakness of Component Object Model in the perspective of implementation.

Answer: -

Security - Only a local server has its address space isolated from that of the client. An in-process server shares the address space and process context of the client and can therefore be less robust in the face of faults or malicious programming.

Granularity. A local server can host multiple instances of its object across many different clients, sharing server state between objects in multiple clients in ways that would be difficult or impossible if implemented as an in-process server, which is simply a DLL loaded into each client.

Compatibility. If you choose to implement an in-process server, you relinquish compatibility with OLE 1, which does not support such servers.

Inability to support links. An in-process server cannot serve as a link source.

Q3. Develop an ActiveX control for displaying an image with scrollbars and to allow the stretching of the image in the window.

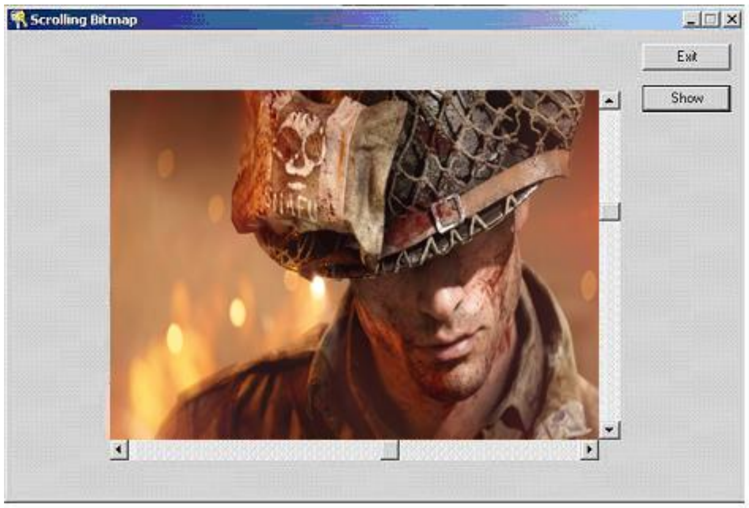
Answer:-

Displaying Images

To load an image at run-time, use the Picture property, and the LoadPicture method:
Set PictureBox.Picture = LoadPicture(strFilePath)
Where PictureBox is the name of your PictureBox control, and strFilePath is the path of the image you want displayed.
The following formats are supported for loading:
Bitmaps (*.bmp;*.dib)
GIF Images (*.gif)
JPEG Images (*.jpg)
MetaFiles (*.wmf;*.emf)
Icons (*.ico;*.cur)

```
        Private Sub Form_Load()
    '// set the path of the image
    sPic = "D: oolbox.gif"
'// display the picture
        Set imgPic.Picture = LoadPicture(sPic)
    With scrPort
'// set the scroll bar properties
.ActiveBar = efstHorizontal
.Max = (imgPic.Width - scrPort.ScaleWidth) Screen.TwipsPerPixelX
.LargeChange = scrPort.ScaleWidth (Screen.TwipsPerPixelX * 2)
.SmallChange = 8
.ActiveBar = efstVertical
.Max = (imgPic.Height - scrPort.ScaleHeight) Screen.TwipsPerPixelY
.LargeChange = scrPort.ScaleHeight (Screen.TwipsPerPixelY * 2)
.SmallChange = 8
End With
'// Move the image to the edge of the control
imgPic.Move 0, 0
End Sub
Private Sub scrPort_Change()
    Dim lL As Long
    Dim lT As Long
    With scrPort
        .ActiveBar = efstHorizontal
        lL = -1 * .Value * Screen.TwipsPerPixelX
        .ActiveBar = efstVertical
        lT = -1 * .Value * Screen.TwipsPerPixelY
    End With
    '// move the image
    With imgPic
                .Move lL, lT, .Width, .Height
        End With
End Sub

Private Sub scrPort_Scroll()
scrPort_Change
End Sub
```

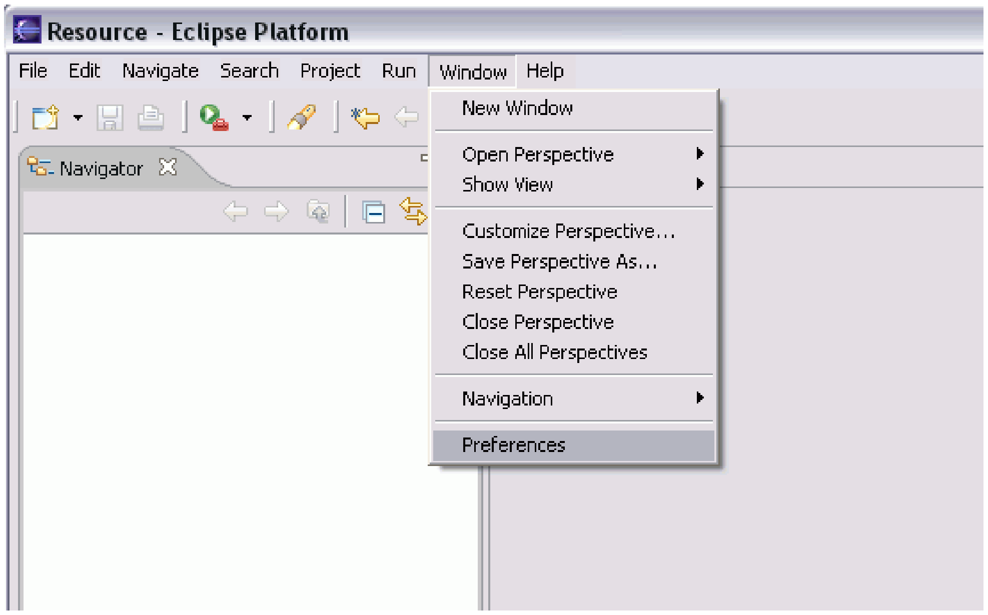
OUTPUT:



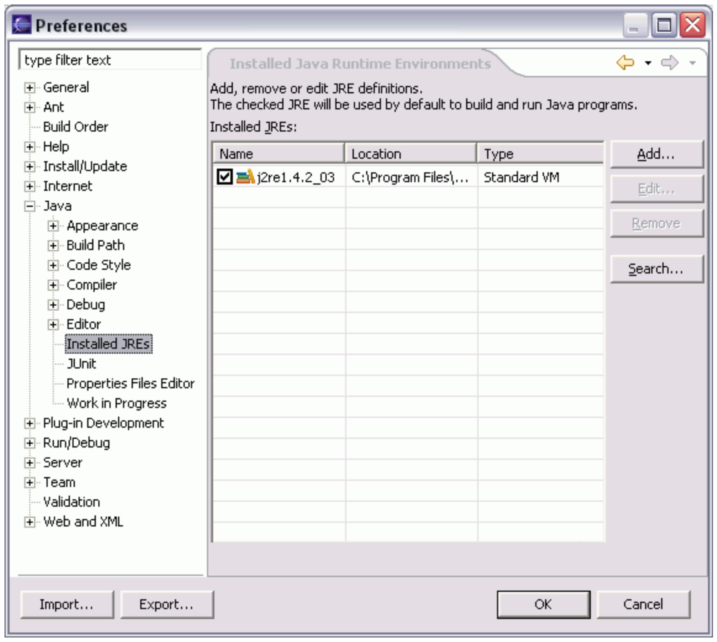
Q4. How will you develop an application using Enterprise Beans?

Answer: -

1. Select Window -> Preferences.



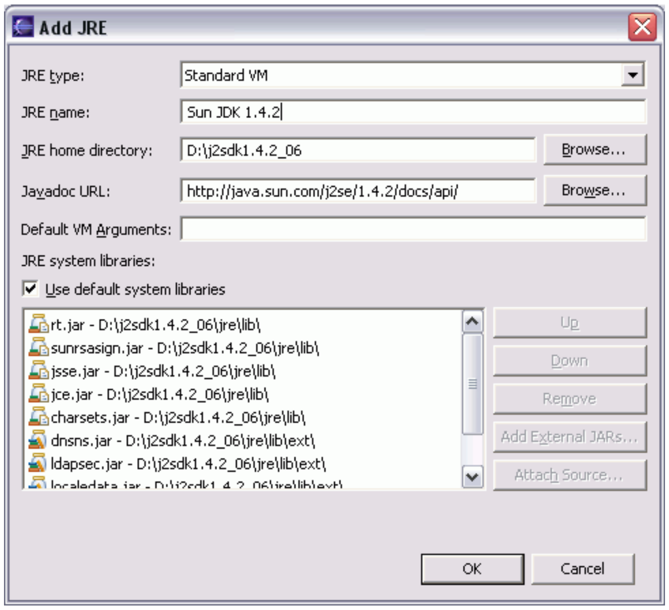
2. Select Java -> Installed JREs from the menu on the left.



Add, remove or edit JRE definitions.
The checked JRE will be used by default to build and run Java programs.
Installed JREs:

Name	Location	Type
☑ j2re1.4.2_03	C:\Program Files\...	Standard VM

3. Select Add... The Add JRE dialog will open. Select Browse for the JRE home directory and choose the location where you installed the JDK 1.4.2. Name the runtime Sun JDK 1.4.2 and click OK.



Add JRE

JRE type: Standard VM

JRE name: Sun JDK 1.4.2

JRE home directory: D:\j2sdk1.4.2_06

Javadoc URL: http://java.sun.com/j2se/1.4.2/docs/api/

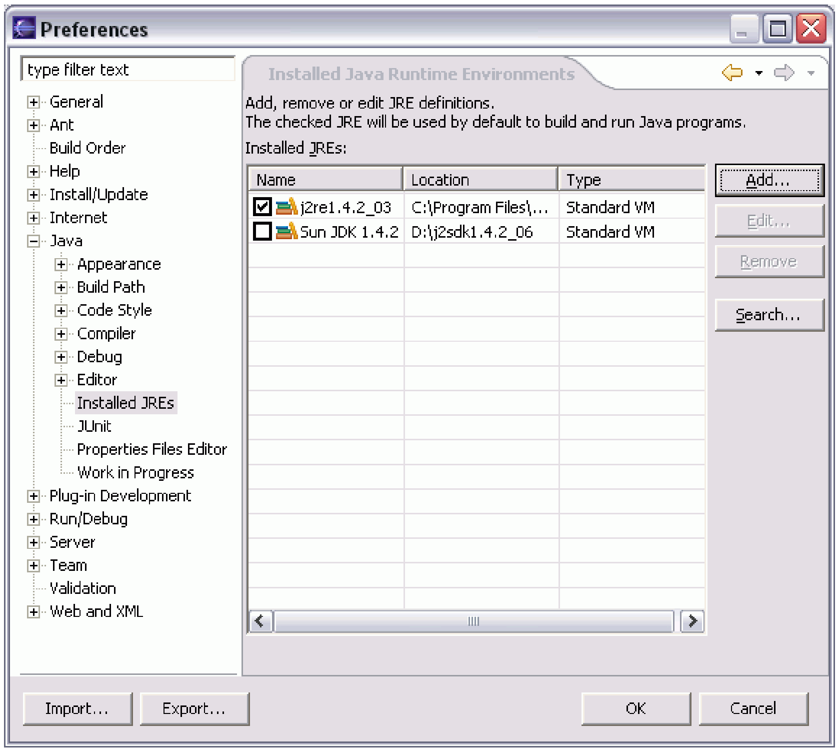
Default VM Arguments:

JRE system libraries:

☑ Use default system libraries

- rt.jar - D:\j2sdk1.4.2_06\jre\lib\
- sunrsasign.jar - D:\j2sdk1.4.2_06\jre\lib\
- jsse.jar - D:\j2sdk1.4.2_06\jre\lib\
- jce.jar - D:\j2sdk1.4.2_06\jre\lib\
- charsets.jar - D:\j2sdk1.4.2_06\jre\lib\
- dnsns.jar - D:\j2sdk1.4.2_06\jre\lib\ext\
- ldapsec.jar - D:\j2sdk1.4.2_06\jre\lib\ext\
- localedata.jar - D:\j2sdk1.4.2_06\jre\lib\ext\

4. The Sun JDK 1.4.2 now shows on the list of installed runtimes. Click OK.



Preferences

type filter text

- General
- Ant
- Build Order
- Help
- Install/Update
- Internet
- Java
  - Appearance
  - Build Path
  - Code Style
  - Compiler
  - Debug
  - Editor
  - Installed JREs
  - JUnit
  - Properties Files Editor
  - Work in Progress
- Plug-in Development
- Run/Debug
- Server
- Team
- Validation
- Web and XML

Installed Java Runtime Environments

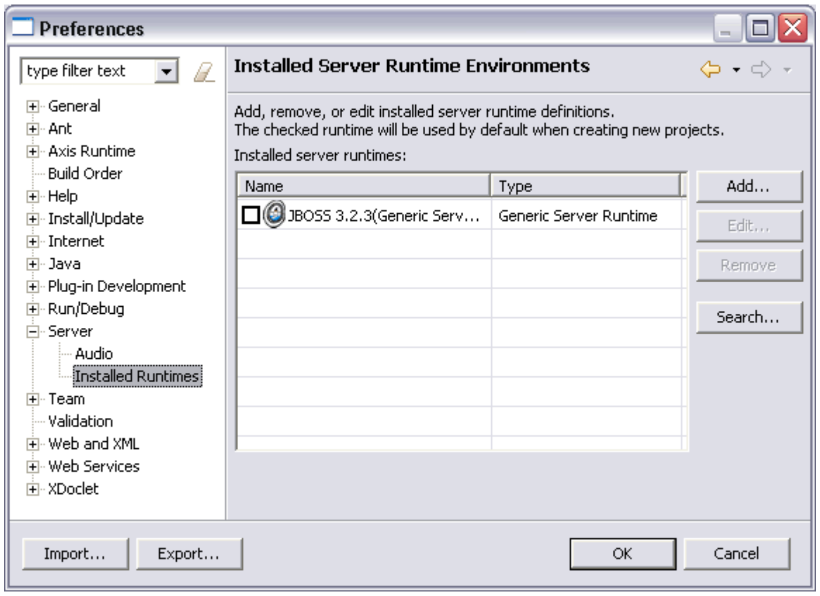
Add, remove or edit JRE definitions.
The checked JRE will be used by default to build and run Java programs.

Installed JREs:

Name	Location	Type
☑ j2re1.4.2_03	C:\Program Files\...	Standard VM
☐ Sun JDK 1.4.2	D:\j2sdk1.4.2_06	Standard VM

5. Select Window -> Preferences.

6. Select Server -> Installed Runtimes from the menu on the left.

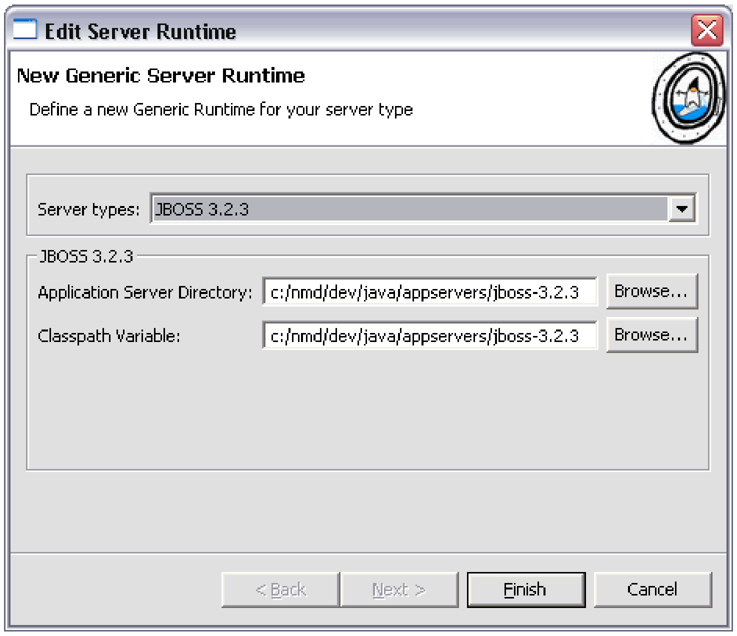


Installed Server Runtime Environments

Add, remove, or edit installed server runtime definitions.
The checked runtime will be used by default when creating new projects.
Installed server runtimes:

Name	Type
☐ JBOSS 3.2.3(Generic Serv...	Generic Server Runtime

7. Click on Add... Select Generic Server Runtime -> JBoss v3.2.3 and click Next. Click Browse and select the location where you installed JBoss. and click Finish.



New Generic Server Runtime

Define a new Generic Runtime for your server type

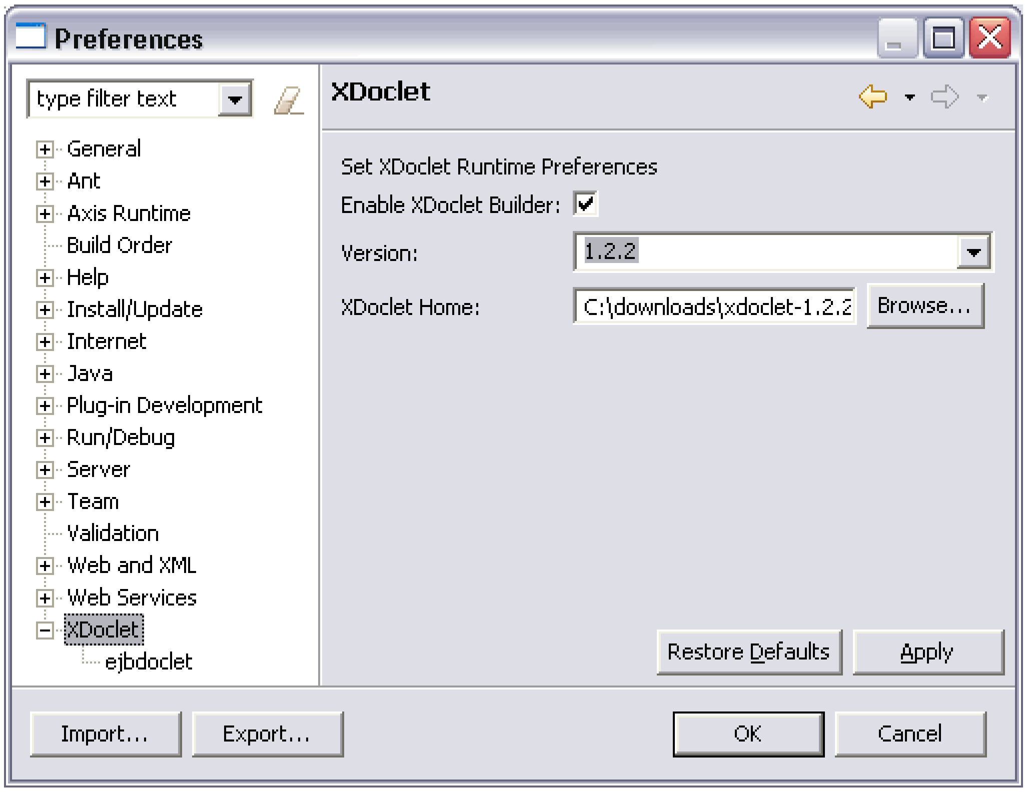
Server types: JBOSS 3.2.3

JBOSS 3.2.3

Application Server Directory: c:/nmd/dev/java/appservers/jboss-3.2.3

Classpath Variable: c:/nmd/dev/java/appservers/jboss-3.2.3

8. Select XDoclet. Make sure the builder item is selected. Click on the Browse... button and choose the directory where you have installed XDoclet. Make sure that you choose the correct version. Click Apply.



XDoclet

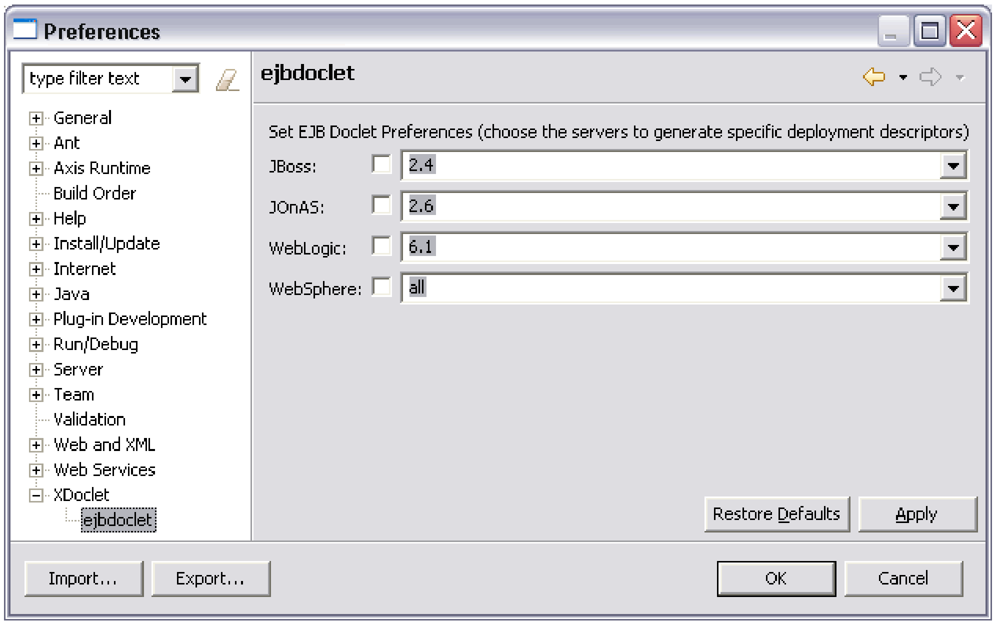
Set XDoclet Runtime Preferences

Enable XDoclet Builder: ✓

Version: 1.2.2

XDoclet Home: C:\downloads\xdoclet-1.2.2 Browse...

9. Select XDoclet->ejbdoclet This is where you can modify the ejbdoclet generation options. You will be able to select specific application servers and their versions. XDoclet will generate deployment descriptors for these application servers only. Make sure nothing is selected. Click Apply.



ejbdoclet

Set EJB Doclet Preferences (choose the servers to generate specific deployment descriptors)

JBoss: ☐ 2.4

JOnAS: ☐ 2.6

WebLogic: ☐ 6.1

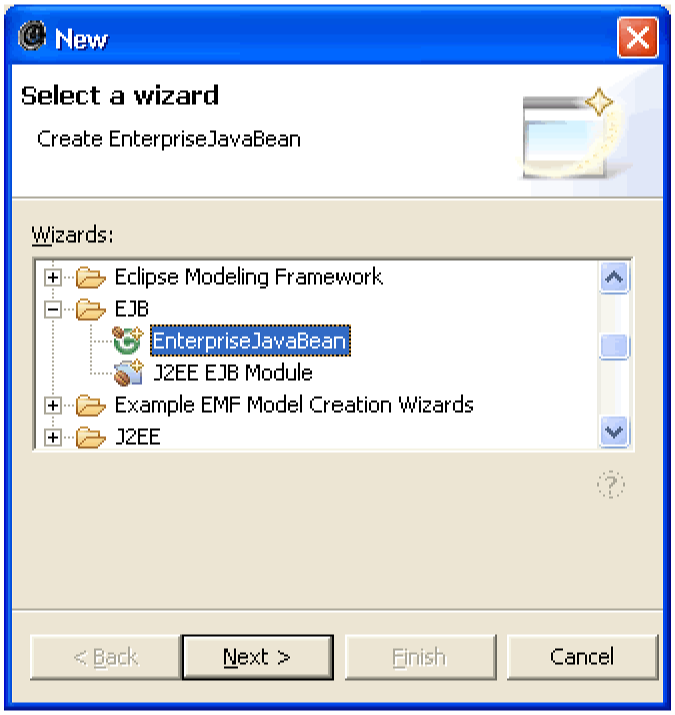
WebSphere: ☐ all

10. Click OK to close the preferences dialog. JDK, XDoclet and JBoss are now configured in Eclipse.

Creating EJB and Web Modules

You will now create flexible projects with EJB and Web modules. In the project with an EJB module you will create a simple Session EJB, and in the project with the web module you will create a client web application to this EJB component.

1. Select File -> New -> Other. Select EJB -> J2EE EJB Module and click Next.



New

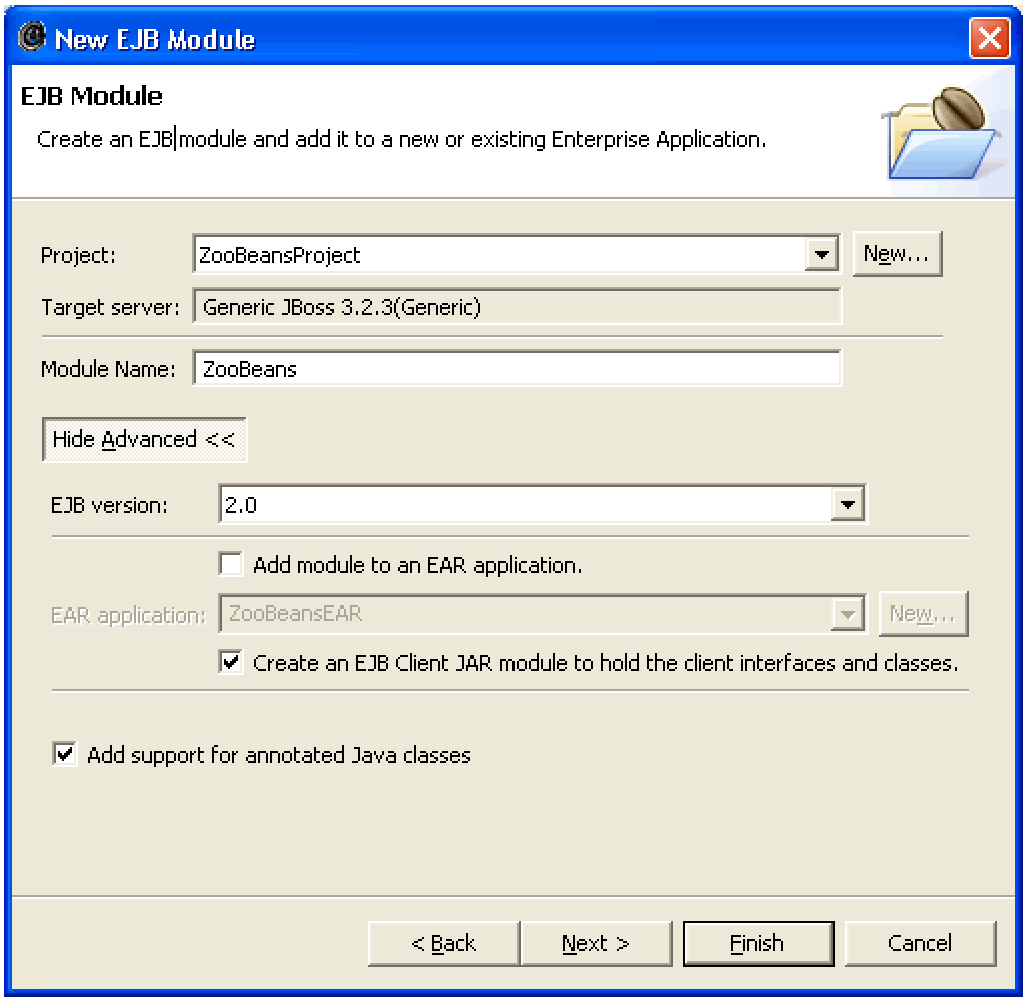
Select a wizard

Create EnterpriseJavaBean

Wizards:

- Eclipse Modeling Framework
- EJB
  - EnterpriseJavaBean
  - J2EE EJB Module
- Example EMF Model Creation Wizards
- J2EE

2. Click New... for the project, name the project ZooBeansProject and choose the server you have defined and click OK. Enter ZooBeans for the module name. Click advanced and deselect Add module to an EAR project. Select Create an EJB Client JAR, make sure the EJB version 2.0 is selected and click Finish to accept the defaults.


New EJB Module

EJB Module

Create an EJB module and add it to a new or existing Enterprise Application.

Project: ZooBeansProject

Target server: Generic JBoss 3.2.3(Generic)

Module Name: ZooBeans

EJB version: 2.0

☐ Add module to an EAR application.

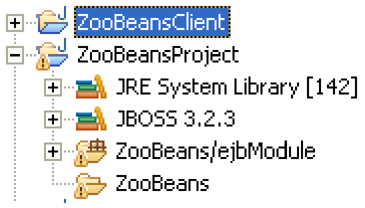
EAR application: ZooBeansEAR

☑ Create an EJB Client JAR module to hold the client interfaces and classes.

☑ Add support for annotated Java classes

3. You will be prompted to switch to the J2EE perspective. Click Yes.

4. The Project Explorer is populated with different types of Web and EJB applications. Your newly created project is located under EJB Projects. Expand your EJB project. The Deployment Descriptor contains information about your EJB project such as the EJBs that your project contains. Expanding the ZooBeans folder will reveal the ejbModule folder. This is where you will store all of your Java files including any EJBs and supporting classes you create. ZooBeansClient project holds the common classes that will be packed into the EJB client jar. We will need this project for the EJB module and the web module.

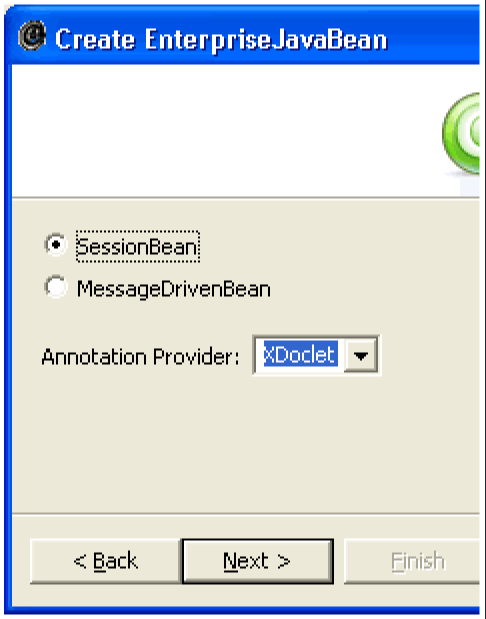


Name-Priyanshu Patidar
Enrollment Number-17100BTCSE01253

Creating a Session EJB

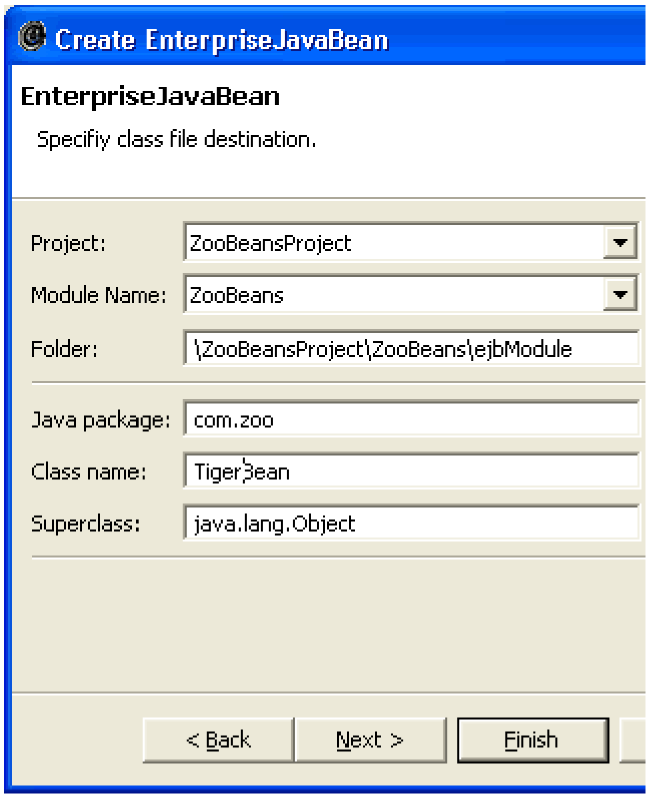
Here we will create the simplest of all EJBs; A stateless session bean.

1. Right click on the ejbModule folder and select New -> Other...-> EJB ->EnterpriseJava Bean. Choose SessionBean in the first tab, in the next tab. Set the package to com.zoo. Next enter the class name; class name must end with a "Bean" so name it the bean TigerBean, Click Next. Click Finish. The TigerBean class will now appear in the ejbModule folder and XDoclet builder will generate Tiger, TigerBean, TigerHome, TigerUtil and other related classes for you. TigerBean is an XDoclet annotated Session Bean and other classes are derived classes. Open TigerBean in the Java editor.



[✓ SessionBean] MessageDrivenBean

Annotation Provider: XDoclet



EnterpriseJavaBean

Specifiy class file destination.

Project: ZooBeansProject

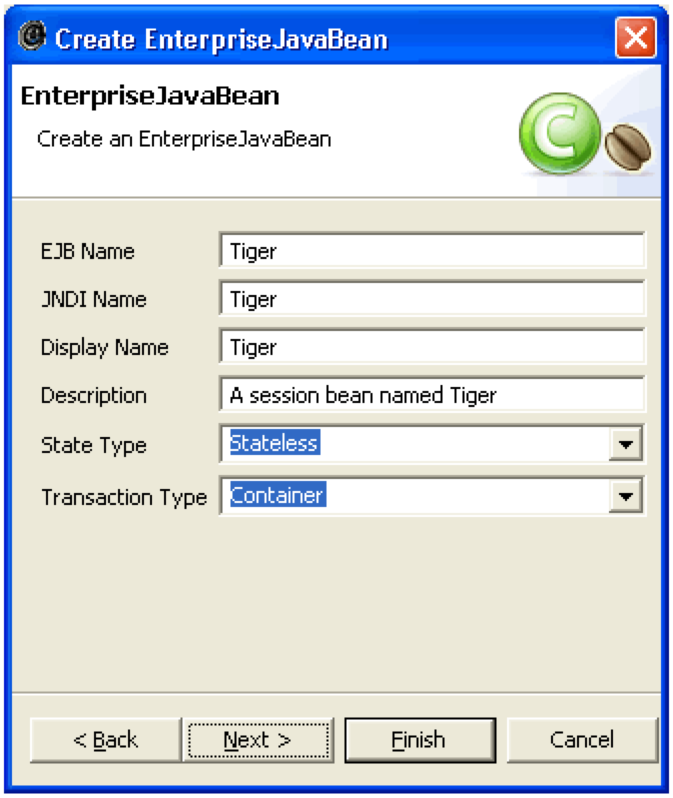
Module Name: ZooBeans

Folder: \ZooBeansProject\ZooBeans\ejbModule

Java package: com.zoo

Class name: TigerBean

Superclass: java.lang.Object



EnterpriseJavaBean

Create an EnterpriseJavaBean

EJB Name	Tiger
JNDI Name	Tiger
Display Name	Tiger
Description	A session bean named Tiger
State Type	Stateless
Transaction Type	Container

2. XDoclet is an extended Javadoc Doclet engine. It's a generic Java tool that lets you create custom Javadoc @tags and based on those @tags generate source code or other files (such as xml-ish deployment descriptors) using a template engine it provides. XDoclet supports a set of common standard tasks such as web.xml or ejb-jar.xml generation. XDoclet uses special JavaDoc @tags to define settings for each component. For example putting a `@ejb.bean name="Tiger" jndi-name="Tiger" type="Stateless"` *in TigerBean.java*

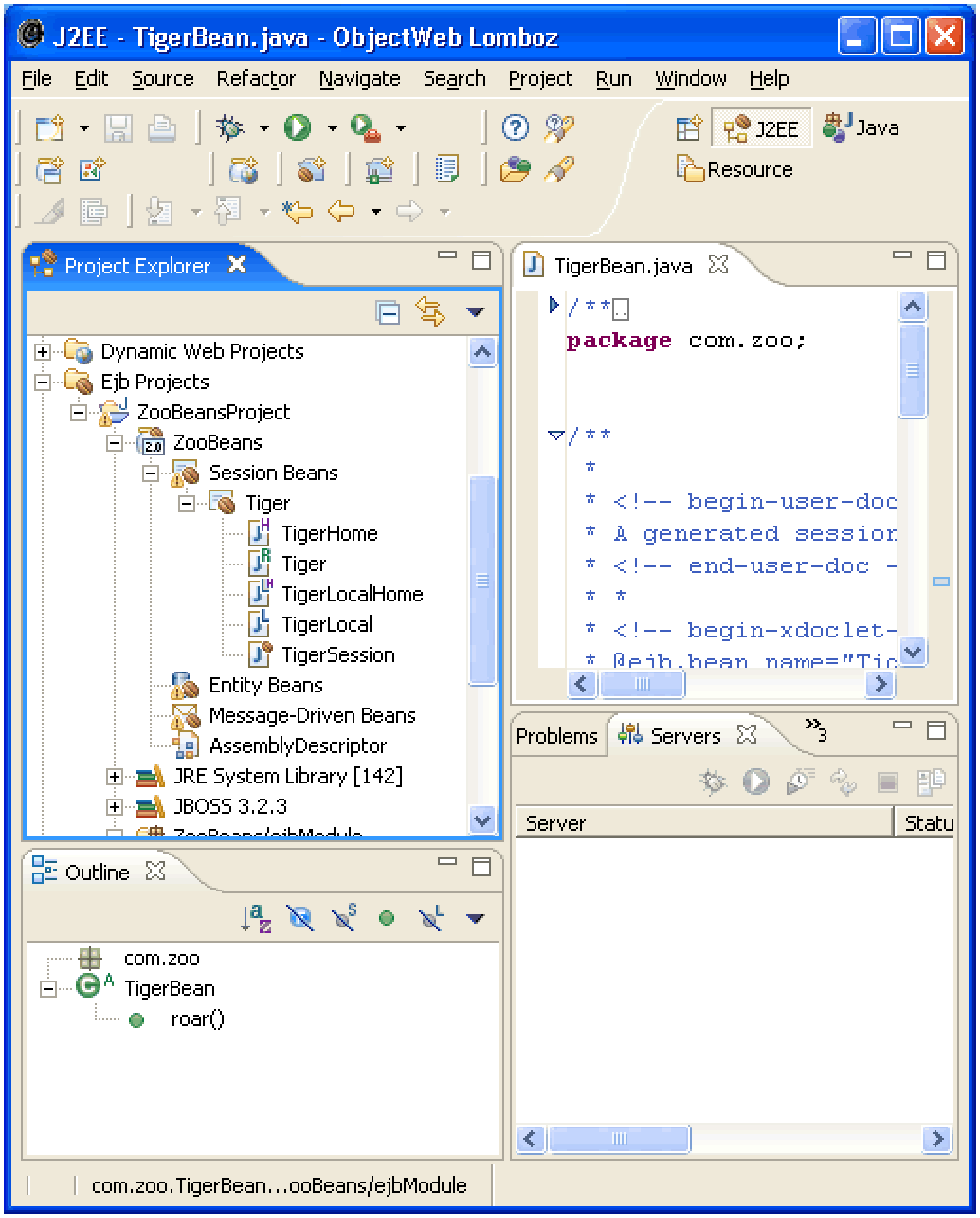
```
/**
 *
 *
 * A generated session bean
 *
 * *
 *
  * @ejb.bean name="Tiger"
  *           description="A session bean named Tiger"
  *           display-name="Tiger"
  *           jndi-name="Tiger"
  *           type="Stateless"
  *           transaction-type="Container"
  *
  *
  * @generated
  */
```

```
public abstract class TigerBean implements javax.ejb.SessionBean {

        /**
         *
         *
         * @ejb.interface-method view-type="remote"
         *
         * @generated
         *
         * //TODO: Must provide implementation for bean method stub
         */
        public String foo(String param) {
                return null;
        }
}
```

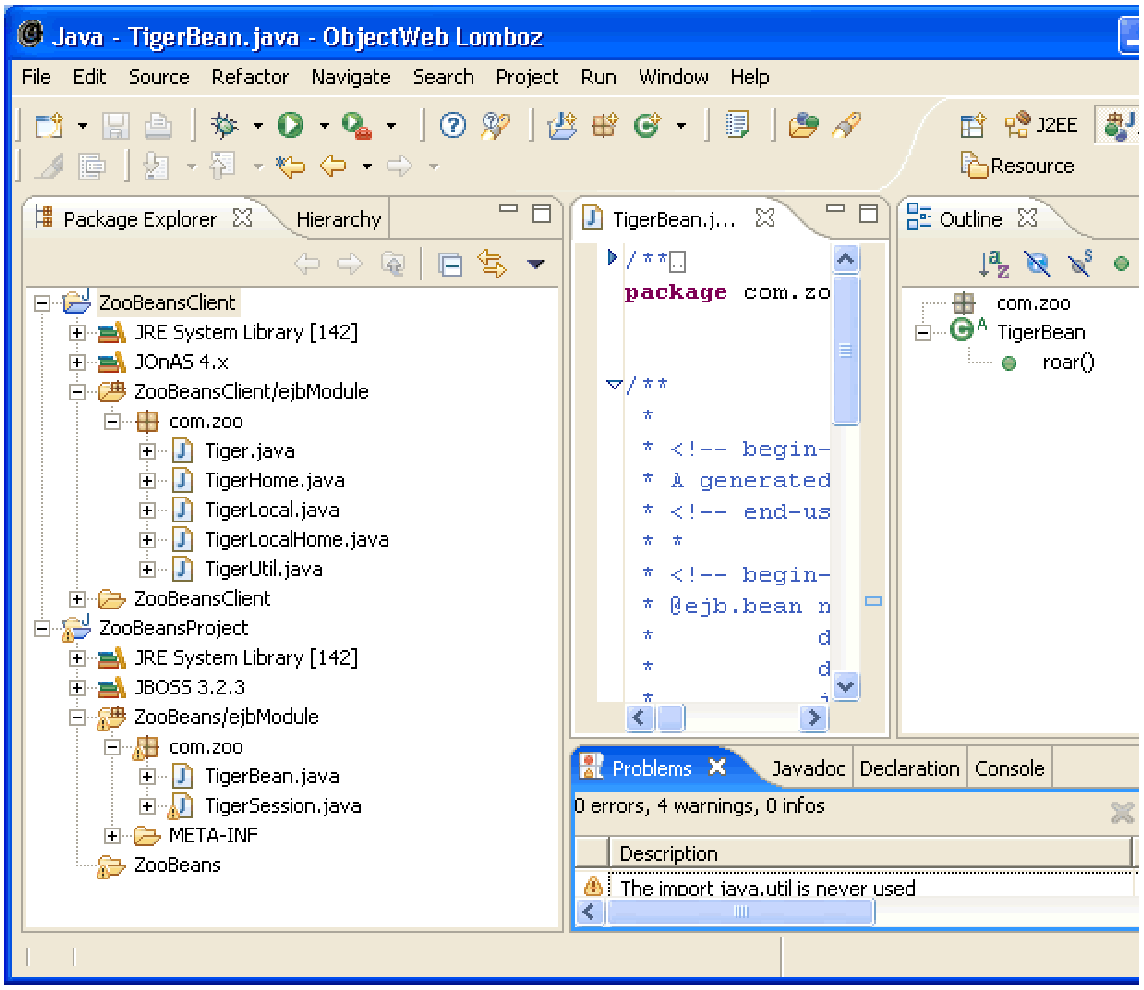
36. TigerBean is the only class you will edit. All others are automatically generated and XDoclet will regenerate them each time you make a change to the TigerBean class. WTP provides code-assist for XDoclet tags. To get help hit crtl-space anywhere inside a xdoclet Javadoc. Edit the TigerBean class and change the foo method to look like this:

```
        /**
         *
         *
         * @ejb.interface-method view-type="remote"
         *
         *
         */
        public String roar() {
                return "Roar!";
        }
```

47. XDoclet builder will start working again and update your classes. At the end your projects will look like the following: The ZooBeansProject that hosts the EJB mmodule will have two classes TigerBean.java and TigerSession.java. These are server side classes. ZooBeansClient project will have the public interfaces such as the Tiger.java and TigerHome.java. These are the classes that will be needed by all clients.



```
/**
package com.zoo;


/**
 *
 * <!-- begin-user-doc
 * A generated sessior
 * <!-- end-user-doc -
 * *
 * <!-- begin-xdoclet-
 * @ejb.bean name="Tic
```



```
/**
package com.zo

/**
 *
 * <!-- begin-
 * A generated
 * <!-- end-us
 * *
 * <!-- begin-
 * @ejb.bean n
 *          d
 *          d
```

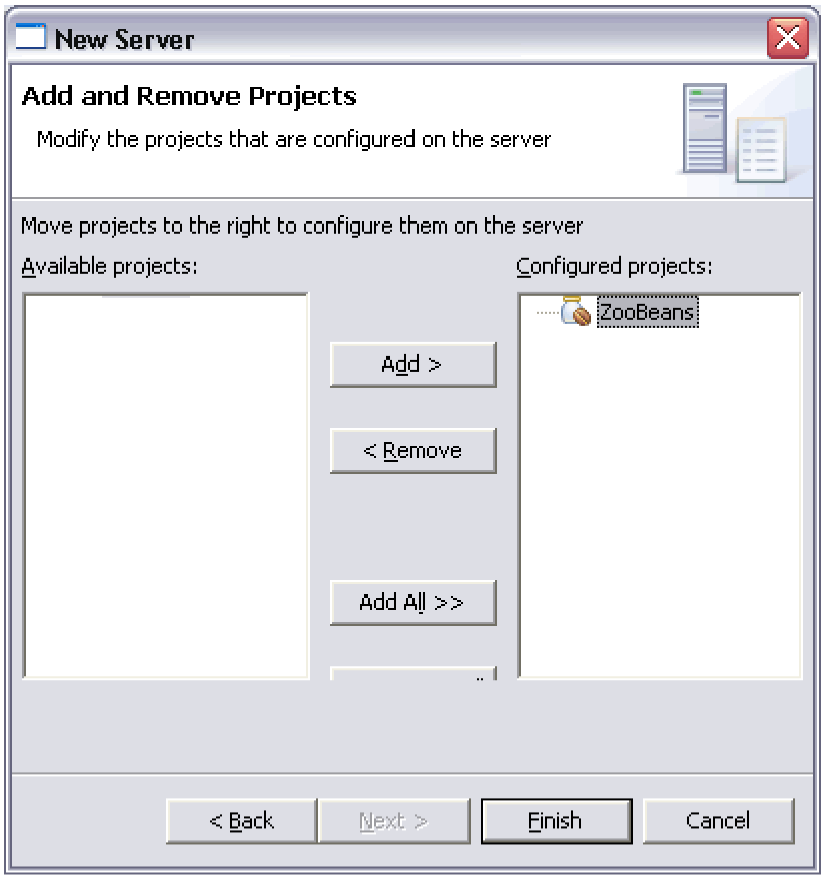
0 errors, 4 warnings, 0 infos

The import java.util is never used

Publishing the EJB to the server

Here we will publish the newly created ZooBeans to a server.

1. Select the server view from the workbench menu Window>Show View>Other>Server>Servers. Right click on the server view window select New -> Server... Select All>Generic Server Support and then Click Next. Keep the setting for JBoss 3.2.3 and click Next.


New Server

Add and Remove Projects

Modify the projects that are configured on the server

Move projects to the right to configure them on the server

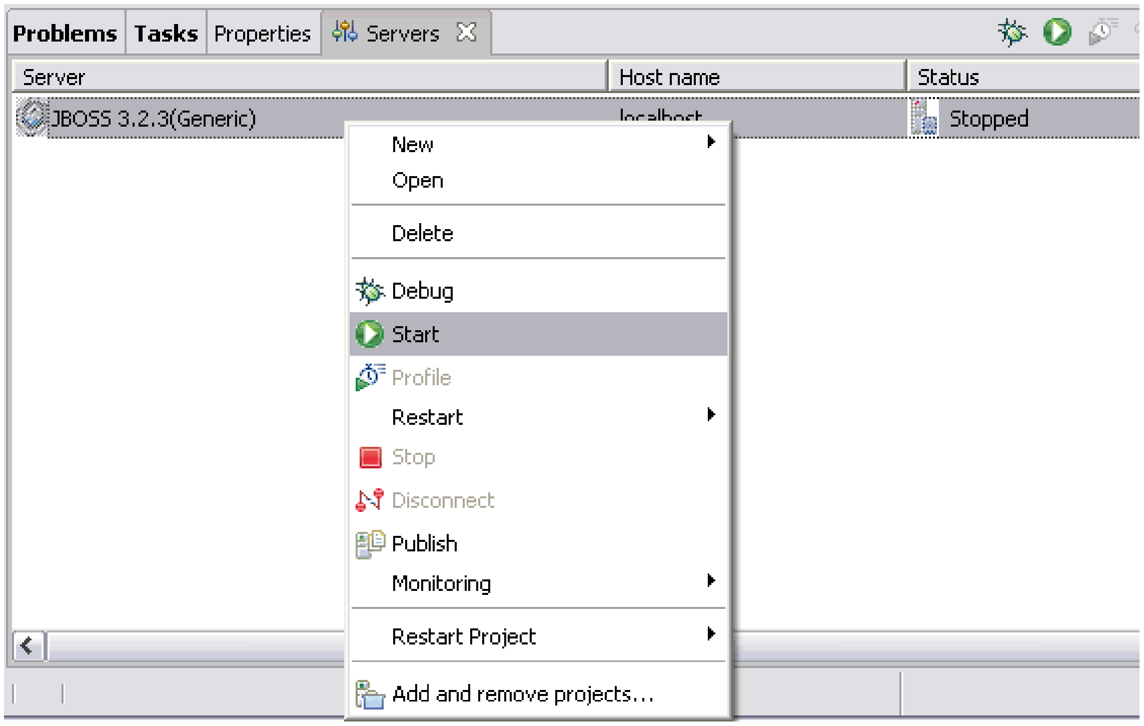
Available projects:

Configured projects:

ZooBeans

Choose the ZooBeans project from the Available Projects list and click Add and then Click Finish.

2. You have setup your server to run your EJB module. Click on the server right click. Choose Start.



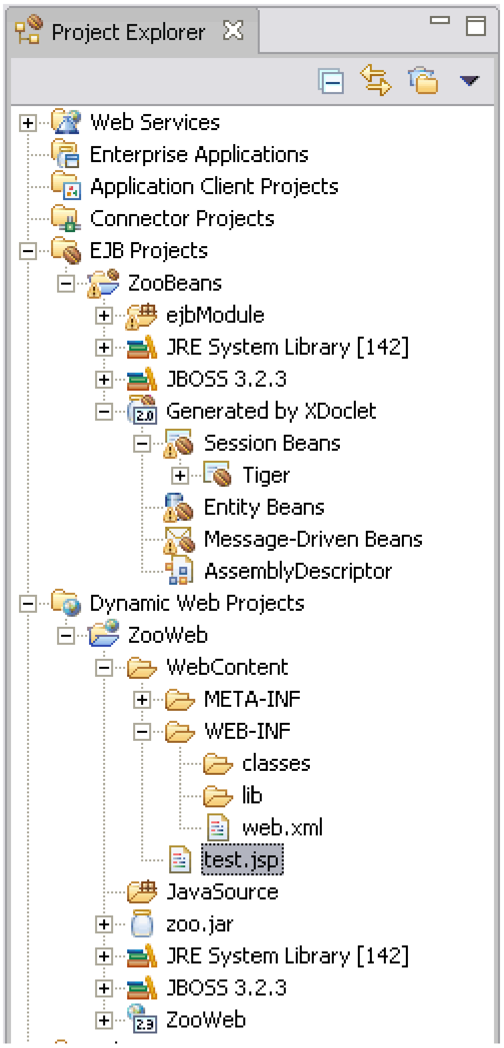
3. You should now observe the console output. JBoss will print a message similar to the following indicating that the EJB Tiger has been deployed successfully.

```
4. 12:46:38,899 INFO  [MainDeployer] Starting deployment of package:
   ZooBeans.jar
5. 12:46:39,249 INFO  [EjbModule] Deploying Tiger
6. 12:46:39,409 INFO  [StatelessSessionInstancePool] Started
   jboss.j2ee:jndiName=Tiger,plugin=pool,service=EJB
7. 12:46:39,409 INFO  [StatelessSessionContainer] Started
   jboss.j2ee:jndiName=Tiger,service=EJB
8. 12:46:39,409 INFO  [EjbModule] Started
   jboss.j2ee:module=ZooBeans.jar,service=EjbModule
9. 12:46:39,409 INFO  [EJBDeployer] Deployed: ZooBeans.jar
10.
```

Testing the Tiger! Building a client web application

Your EJB has now been defined with your EJB Project and it has been deployed. It is ready to be used. You can now build a test client to make the Tiger EJB Roar! To make this create a Flexible Project with a Web Module named ZooWebProject. Use the same server (JBoss) for this project too. You can follow the step described in the tutorials Building a School Schedule Web Application or Building and Running a Web Application if you are not familiar with the process.

1. When you have created the ZooWebProject. You will end up with a project that looks like the following:

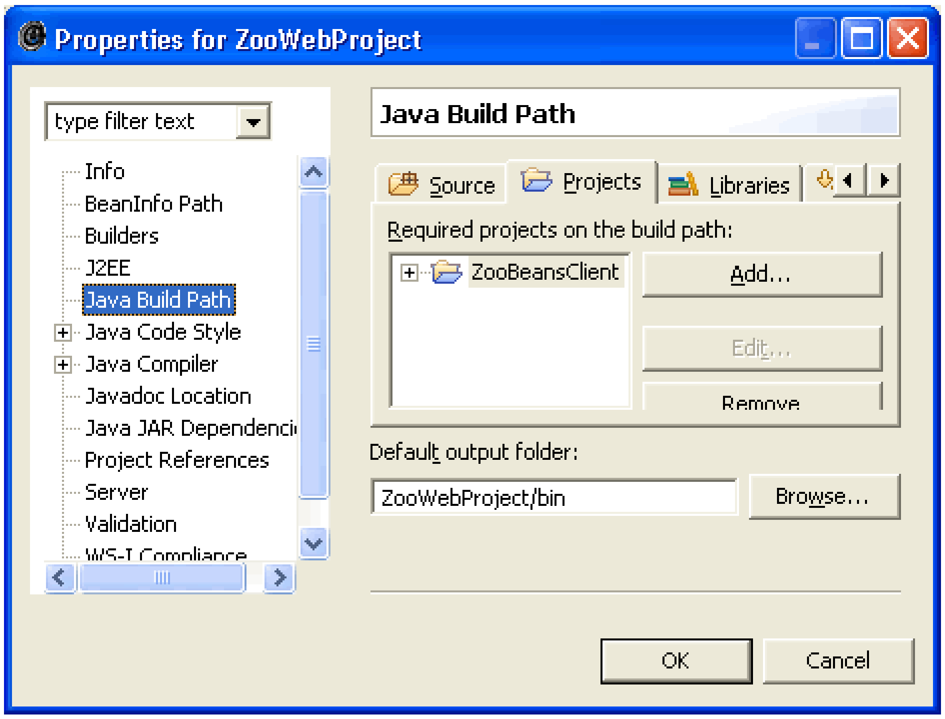


This Web module will be client to the EJB. Therefore it needs to know about the EJB module we have created. To do this we will set the project build path and use the Flexible projects to dynamically include the EJB client jar with our web module.

Using Flexible projects to link the web module with the an EJB Client module

We will need to access the EJB interface types such as Tiger and TigerHome in client applications. These classes are in the ZooBeansClient project, and are packaged in an EJB client jar (ZooBeansClient.jar). We need to add the client classes to the build path and inlcude the client.jar in the web module when it is deployed to a server.

1. In the Project or Package Explorer, right click on the ZooBeansWeb project and Choose Properties... In the Java buildpath, add the ZooBeansClient project to the project references. This will help allows us to compile against the latest ejb client classes in this web project.



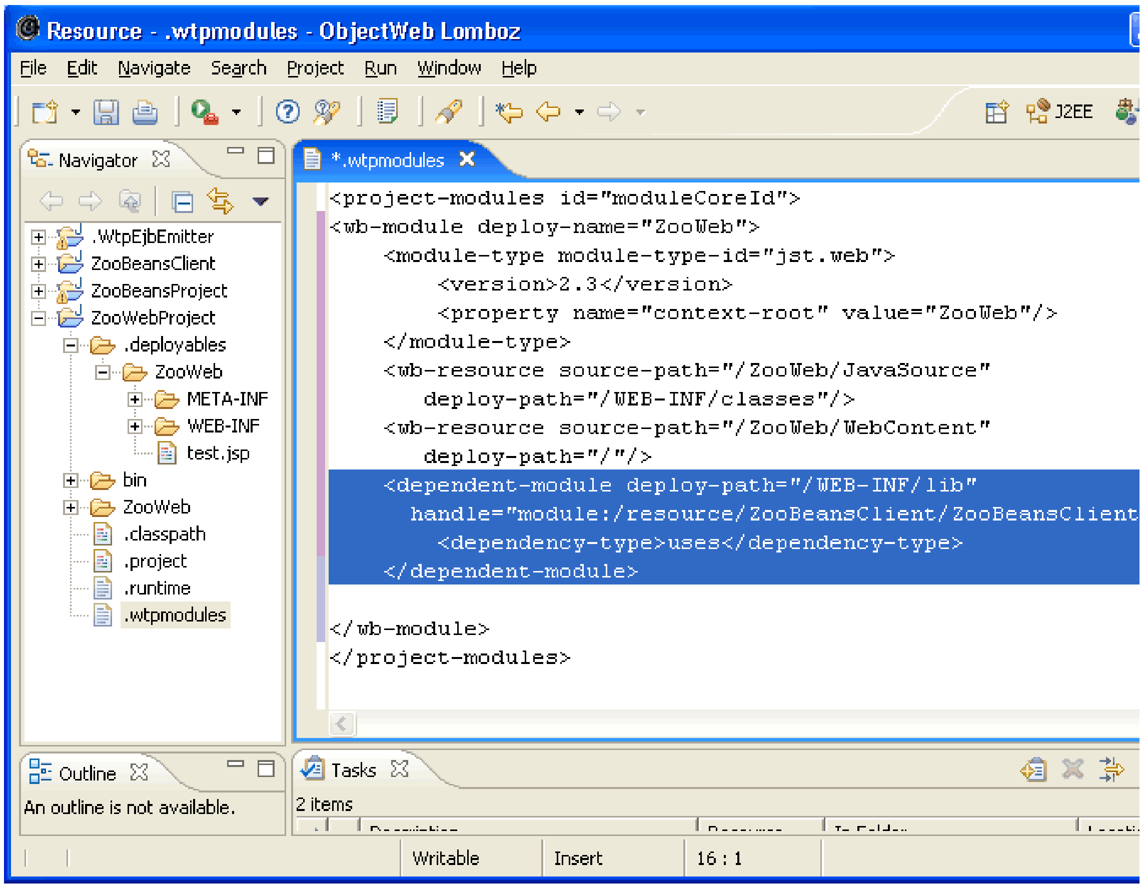
Required projects on the build path:
ZooBeansClient

Default output folder:
ZooWebProject/bin

2. Now we have to add a dependent module to the ZooWeb web module. We will use the flexible project support to do this. Currently, WTP does not have a user interface to add dependent modules, so we will edit a file named ".wtpmodules", which can be found at the root of each flexible project. However, this is a hidden file so too see it, one must use the "Resource Perspective". To go to the resource perspective go Window>Open Perspective>Other and choose resource. Open the .wtpmodules file and edit the contents to add the following:

```

    <dependent-module deploy-path="/WEB-INF/lib"
      handle="module:/resource/ZooBeansClient/ZooBeansClient">
        <dependency-type>uses</dependency-type>
    </dependent-module>
```

Here we have asked the flexible project support to include the ejb client jar inside the WEB-INF/lib folder of the web module during runtime deployment. Flexible project support will do the tasks of building a new jar and including it with our web module each time the ejb changes.



```
<project-modules id="moduleCoreId">
<wb-module deploy-name="ZooWeb">
    <module-type module-type-id="jst.web">
        <version>2.3</version>
        <property name="context-root" value="ZooWeb"/>
    </module-type>
    <wb-resource source-path="/ZooWeb/JavaSource"
       deploy-path="/WEB-INF/classes"/>
    <wb-resource source-path="/ZooWeb/WebContent"
       deploy-path="/"/>
    <dependent-module deploy-path="/WEB-INF/lib"
      handle="module:/resource/ZooBeansClient/ZooBeansClient
        <dependency-type>uses</dependency-type>
    </dependent-module>

</wb-module>
</project-modules>
```

8. Now Create a file named test.jsp
   Open test.jsp in the JSP source page editor.

9. Add the following lines to test.jsp

```
<html>
<head>
<title>Insert title here</title>
</head>
<body>
<%
   com.zoo.Tiger tiger = null;
   try{
           com.zoo.TigerHome home = com.zoo.TigerUtil.getHome();
           tiger = home.create();

   }catch(Exception exception)
   {

   }


%>
<b><%= tiger.roar() %></b>
</body>
</html>
```

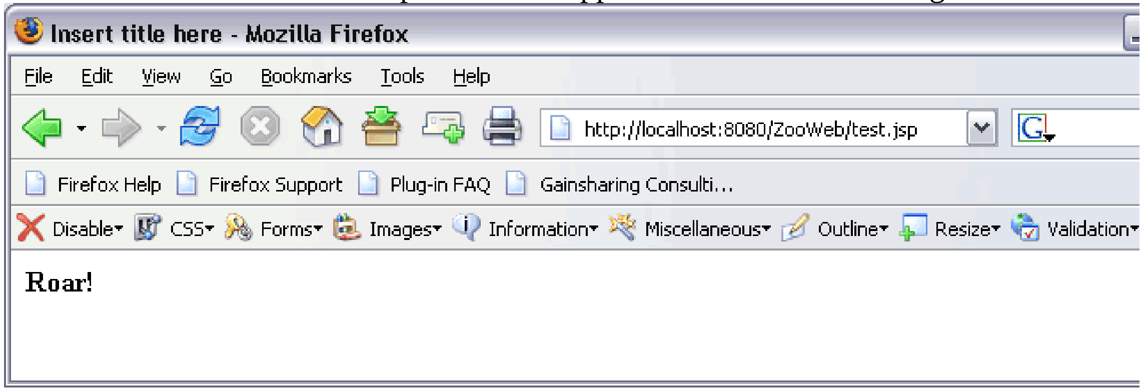
Name-Priyanshu Patidar
Enrollment Number-17100BTCSE01253

31. Save the test file.

Running And Testing The Web Application

Your web application is now complete. Now it is time to take it for a spin.

1. Right click on test.jsp and select Run As -> Run on Server...
2. A Web browser window will open with the application. Beware of the tiger!.



Roar!

3. Experiment adding more methods and ejbs.