Fundamentals of Java Enterprise Components

Session: 10

Enterprise JavaBeans

Objectives

- Explain enterprise beans
- Use variants of enterprise beans
- Describe different ways through which the enterprise beans can be accessed
- Describe the lifecycle of the beans
- Create and deploy a sample enterprise bean using NetBeans IDE

Enterprise Beans

Enterprise beans are server-side components, which implement the business logic of the application.

Enable modularized development.

The bean components have scalability, transaction management, and security authorization implemented.

The enterprise beans are placed in a container of the application and communicate through interfaces with the application server.

Entity beans also interact with persistent storage of the application.

Types of Beans

There are three types of beans:

- Entity beans (deprecated)
- Session beans
- Message driven beans

Entity beans are deprecated in the current versions of EJB and are replaced by the Persistence API.

Session beans are used to handle a specific client of the application.

Message driven beans are used to process asynchronous messages of the application.

Session Beans

- Session bean is invoked by the client when it has to access the application server.
- The client communicates with the server by invoking session bean methods.
- The session bean methods invoke application components on the application server.
- Session beans are not persistent and cannot survive server crashes.
- Given here is an example of a session bean with methods.

```
import javax.ejb.EJBObject;
import java.rmi.RemoteException;
public interface TravelAgent
extends EJBObject {
public void
reserveTicket(Customer cust,
Train no ri, Date journey_date ,
ResClass c) throws
RemoteException;

public TrainInfo
availability(ResClass loc, Date
journey_date) throws
RemoteException;
}
```

Types of Session Beans 1-4

There are three types of session beans

- Stateful
- Stateless
- Singleton

A stateful session bean is an extension to the server. It is created by the client and maintains the state of the current session. A developer uses stateful beans in an application when:

- Methods are invoked based on the state of the bean
- Certain information has to remain constant between method invocations
- The bean has to interact with various components of the application

Stateless session beans do not maintain any instance variables of the session.

The bean can be identified as stateful or stateless with the @Stateful and @Stateless annotations respectively.

Types of Session Beans 2-4

Following code demonstrates a stateless session bean on the server-side of the application:

```
package Converter;
import java.lang.Float;
import javax.ejb.Stateless;
import javax.ejb.LocalBean;
@Stateless
@LocalBean
public class ConverterBean {
private Float euroRate = new Float("83.4093016");
public Float dollarToEuro(Float dollars) {
Float result = (dollars*euroRate);
return result;
}
```

Web services when implemented with stateless beans are more efficient.

Types of Session Beans 3-4

Following code demonstrates a bean on the client-side:

```
package Converter;
import javax.ejb.Stateless;
import javax.ejb.LocalBean;
import java.lang.Float;
@Stateless
@LocalBean
public class Client1 {
public static void main(String args[]){
   ConverterBean conObj = new ConverterBean();
   Float res;
   Float input;
   System.out.println("Enter the value");
   input = Float.parseFloat("20.00");
   res = conObj.dollarToEuro(input);
    System.out.println(res);
        }
}
```

Types of Session Beans 4-4

A singleton session bean is created only once throughout the application and exists throughout the lifecycle of the application.

The state of a singleton session bean is expected to be retained across method invocations but not across server crashes or shut down.

Used to implement Web services end points.

A developer can use a singleton session bean when:

- Certain tasks need to be executed when the application starts up and when the application shuts down
- Certain information needs to be shared across methods of the application
- An enterprise bean needs to be accessed by multiple threads concurrently

Message Driven Beans

- Process asynchronous messages such as JMS messages in the application.
- Message driven beans are placed in a container.
- It is a stateless, transaction aware component on the server-side.
- To write message driven beans, javax.jms.message package has to be imported.
- Message driven beans cannot be accessed through interfaces.
- A single message driven bean can process messages from multiple clients.
- onMessage() method of the message driven bean processes a message received.
- The received message can be Stream, Map, Text, Object, or Bytes.

Accessing Enterprise Beans

Message driven beans cannot be accessed explicitly through interface methods. They are invoked only by messages.

Session beans can be accessed through the interface provided.

A client can access session beans in the following two ways:

- No-interface view
- Business interface

In no-interface view, the enterprise bean is accessed through the public methods of the enterprise class.

In order to use no-interface view, the client and target bean should be packaged in the same application.

Business interface comprises the business methods of the enterprise bean.

Using Session Beans on the Client Side

Mechanisms used for accessing the components of the application are as follows:

- Java Naming and Directory Interface (JNDI)
- Java annotations
- Dependency injection

JNDI provides explicit look up through a global syntax for Java EE and Java SE components.

JNDI is used for identifying different components of the applications through JNDI names.

Portable JNDI Syntax 1-3

- Provides a mechanism of binding an object to a name.
- Provides an interface to lookup the directory and locate the required component.
- Keeps track of any changes made to the names of the components.
- JNDI defines namespaces and naming conventions to identify different components of the application.
- JNDI defines three namespaces - java:global, java:module, java:app.
- JNDI hierarchically organizes the objects and binds the objects with their respective references in the directory.

Portable JNDI Syntax 2-3

java:global namespace is used to locate remote enterprise beans through JNDI lookups. Following is the format for accessing an enterprise bean in global namespace:

- `java:global[/application name]/module name /enterprise bean name[/interface name ]`

If the client and the enterprise bean are collocated, java:module namespace is used. Following is the format of the java:module namespace:

- `java:module/enterprise bean name/[interface name]`

In order to access Java enterprise beans located within the same application, java:app namespace is used. Following is the format of a java:app address:

- `java:app[/module name]/enterprise bean name [/interface name]`

Portable JNDI Syntax 3-3

This decision of placing the application components is based on the following factors:

- The nature of the design in linking the components
- The type of clients
- The nature of the application if it is distributed

The design choice of locating the components remotely or locally is to be made considering factors such as network latency and other hardware related issues.

The enterprise beans can be accessed both remotely and locally through remote and local interfaces. This can be defined through @Remote and @Local annotations.

Different Types of Clients Accessing Enterprise Beans 1-4

There are primarily three types of clients:

- Local clients
- Remote clients
- Web clients

A client is termed to be a local client when it runs in the same application as the bean and the client knows the absolute location of the enterprise bean.

A local client can access the bean either through no-interface view or through business interface view.

Different Types of Clients Accessing Enterprise Beans 2-4

Local business interface can be defined through one of the following ways:

A bean can be accessed in no-interface view by providing annotations

- `@Session`
- `public class Bean_Impl{….}`

A no-interface view can also be defined by annotating the business interface as local

- `@LocalBean`
- `public class Bean_Imple{…..}`

By specifying the interface class using the @Local annotation and then implementing the interface through the bean class

- `@LocalBean(ExampleBean.class)`
- `Public class Bean_Impl implements ExampleBean{……..}`

Different Types of Clients Accessing Enterprise Beans 3-4

Dependency injection and JNDI lookup can also be used to access the enterprise beans.

Local bean reference can be obtained through @EJB annotation as shown:

```
@EJB
Business_interface_name name;
```

lookup() method of javax.naming.InitialContext can be used to obtain a reference of local business interface through JNDI lookup.

```
Local_Business_interface example = (Local_Business_interface)
InitialContext.lookup("java:module/Local_Business_interface");
```

Different Types of Clients Accessing Enterprise Beans 4-4

A remote client either runs on a different machine or on a different JVM than that of the enterprise bean.

The enterprise bean must have a business interface so that remote clients can access it.

There are two ways of defining the business interface through @Remote annotation:

```
• @Remote
• public interface Interface_Name { ... }
• or
• @Remote(Interface_Name.class)
• public class Bean1 implements Int_Name { ... }
```

Client access to the business interface located remotely can also be processed using dependency injection and JNDI lookup.

Web Services

A Web service is implemented through stateless session bean.

A Web service is invoked through protocols such as Simple Object Access Protocol (SOAP), HyperTextTransferProtocol (HTTP), and Web Service Definition Language (WSDL).

Web service clients cannot invoke message driven beans.

@WebMethod annotation is used to expose the methods of the bean class to the Web service clients.

Naming Conventions of Enterprise Beans 1-3

An enterprise bean comprises different components that work together to accomplish a task.

Bean class implements all the business methods expected to be provided by the bean application.

Given here is an example of a bean class.

```
class Train_ResBean implements
Train_Res{
String reserve_ticket(){
// implementation of the method
}
void cancel_ticket(){
//implementation of the method
}
void get_status(){
//implementation
}
```

Naming Conventions of Enterprise Beans 2-3

Business interface declares the business methods of the bean class that it wishes to expose.

Following code shows the business interface for the Train_ResBean class:

```
public interface Train_Res{
String reserve_ticket();
void cancel_ticket();
void get_status();
}
```

Naming Conventions of Enterprise Beans 3-3

Following table shows the naming conventions used with different components of the enterprise bean:

Component	Convention	Example
Enterprise bean name (DD)	<name >EJB	Train_ResEJB
EJB JAR display name (DD)	<name >JAR	Train_ResJAR
Enterprise bean class	<name >Bean	Train_ResBean
Business interface	<name>	Train_Res

Lifecycle of Enterprise Beans 1-2

Enterprise beans go through different states during their lifecycle. These states are different for stateless, stateful, and singleton beans.

Following are various stages in stateful session beans:

- Instantiation
- Dependencies injected
- Activated
- Passivated
- Removed

The transition among different stages of the lifecycle of the stateful session bean is done through various methods such as:

- ejbActivate()
- ejbPassivate()
- ejbRemove()
- setSessionContext() and newInstance() are used to instantiate the bean

Lifecycle of Enterprise Beans 2-2

Stateless session beans do not keep track of the properties pertaining to the session. Following are the states in the lifecycle of the stateless session bean:

- Instantiated
- Dependencies injected
- Activated
- Removed

The singleton session bean is created only once in the application. Following are different states in the lifecycle of a singleton session bean:

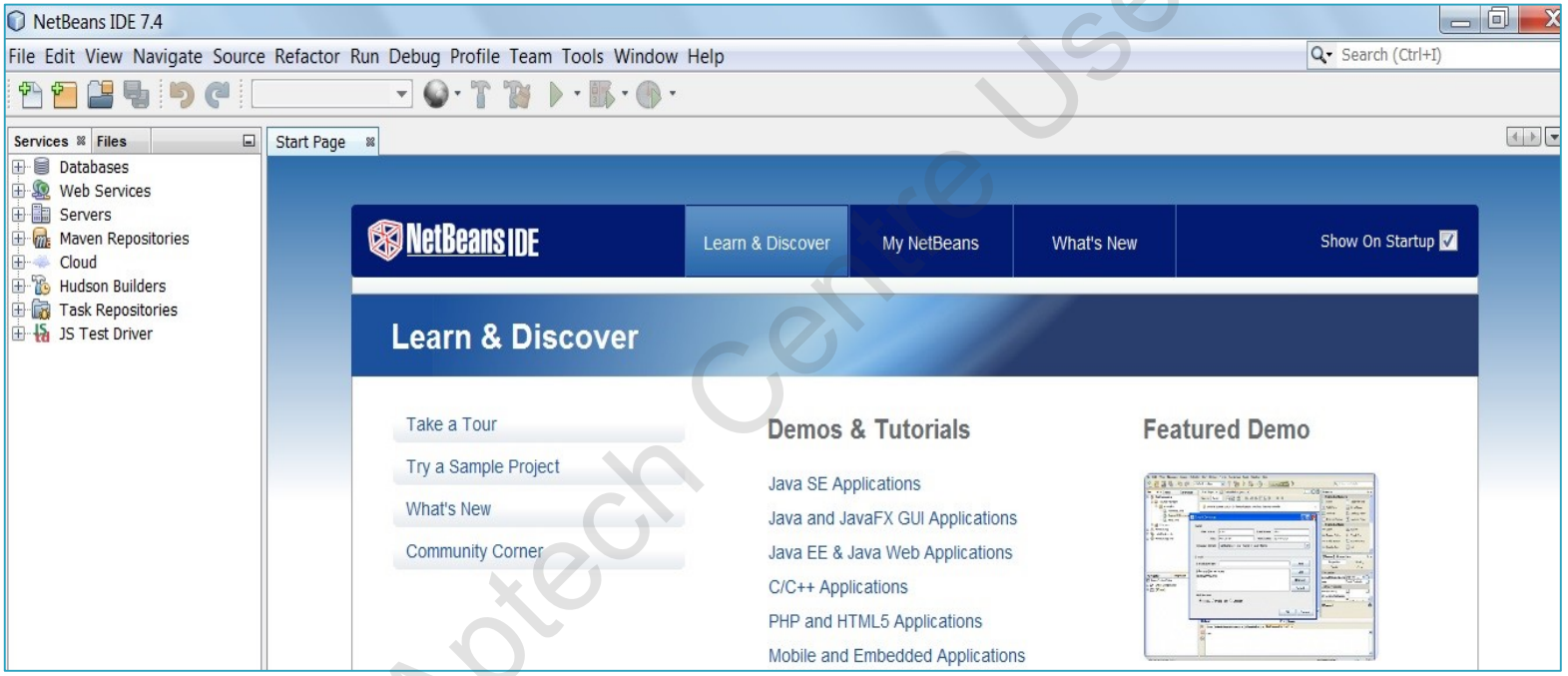
- Instantiated
- Dependencies injected
- Removed

Following are different states in the lifecycle of a message driven bean:

- Instantiated
- Dependencies injected
- Activated
- Removed

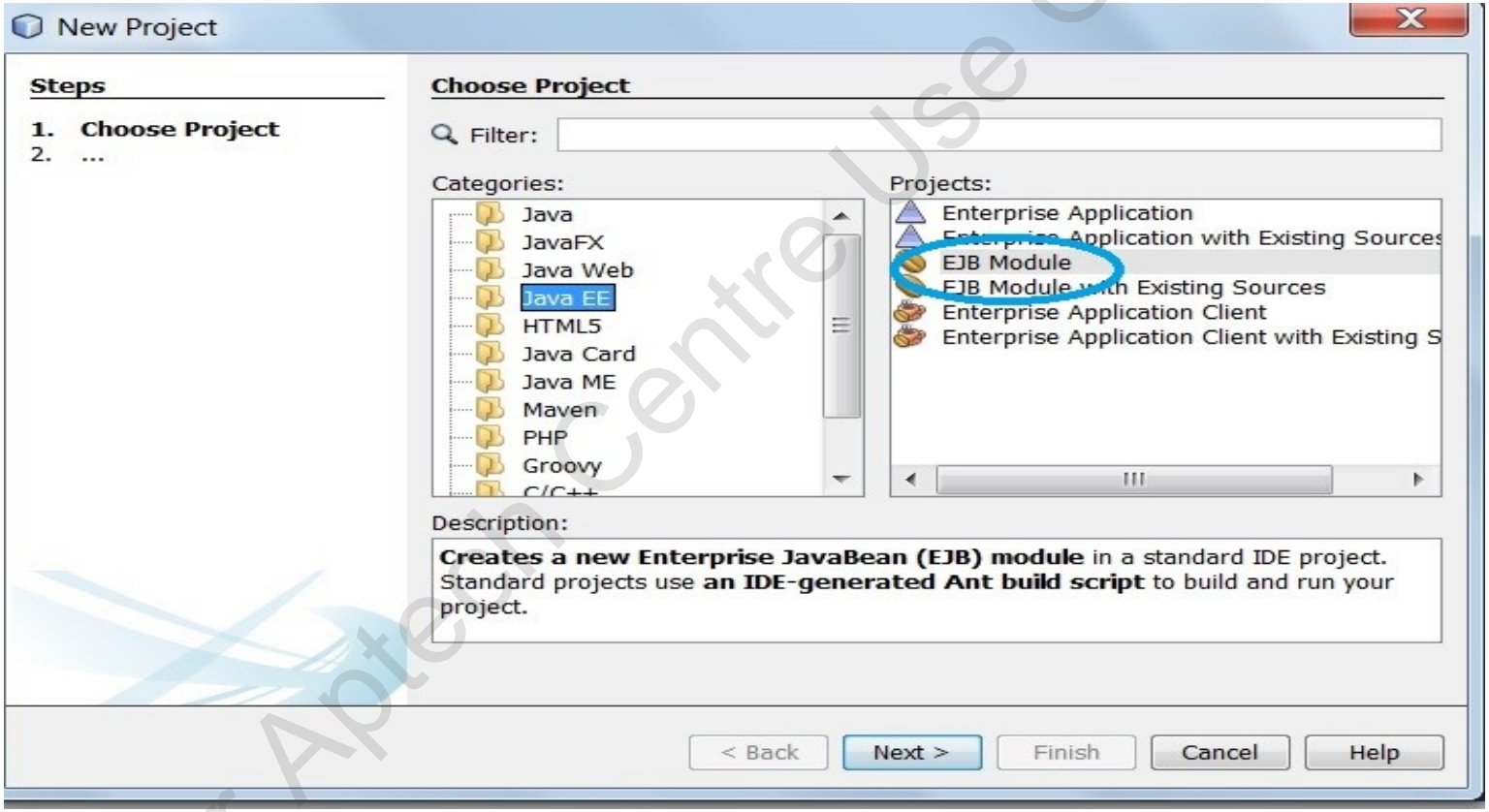
Applying Enterprise Beans 1-10

Let's create a simple bean using Netbeans IDE. Following figure shows the interface of Netbeans IDE:



Applying Enterprise Beans 2-10

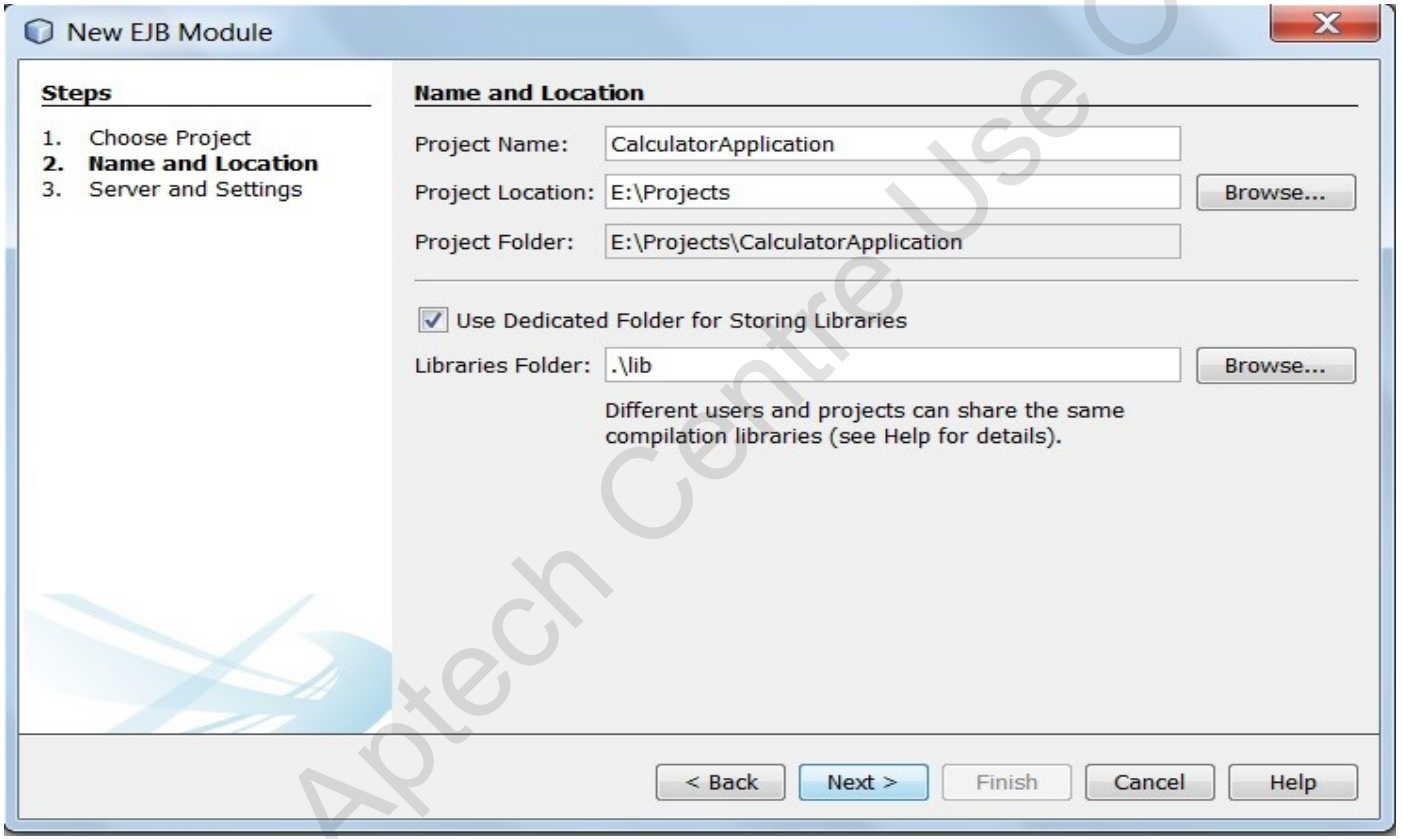
Create a new project by selecting File → New → Java EE → EJB module in the New Project dialog box as shown in the following figure:



Click Next. It leads you to another window where you can choose the project name.

Applying Enterprise Beans 3-10

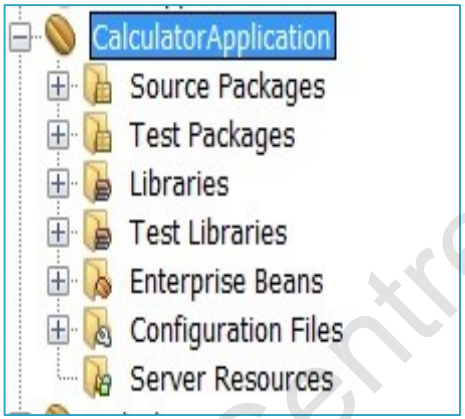
Choose appropriate project name as shown in the following figure:



After specifying the project name, select the server you wish to use for the application and click Finish.

Applying Enterprise Beans 4-10

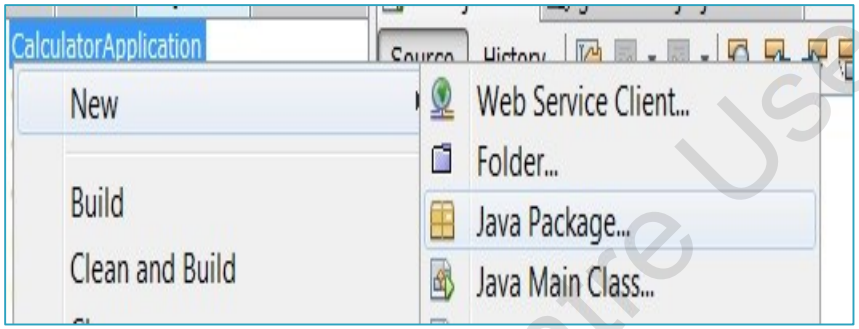
This will create the project with various sub-folders as shown in the following figure:



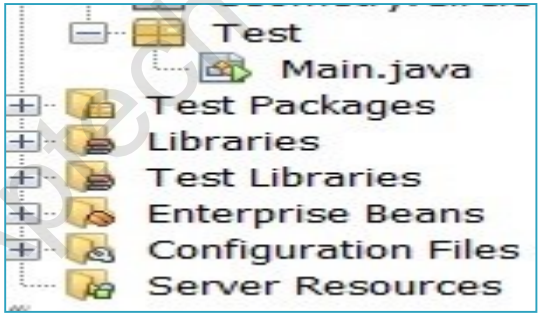
The project is now created with various sub-folders. Source Packages is the folder in which the beans are stored.

Applying Enterprise Beans 5-10

Right-click the Source Packages folder and create a new package within the Source Packages folder as shown in the following figure:



Select New → Java Package. Create a package and give it an appropriate name. Here, the package is named Test as shown in the following figure:



The Main.java class has been added to the Test package. This class will be the client that will invoke the bean methods.

Applying Enterprise Beans 6-10

Now, you can add Calculator interface and CalculatorImplBean to the application.

Following code demonstrates the Calculator interface:

```
package Test;
import javax.ejb.Remote;
@Remote
public interface Calculator {
public String sayHello(String name);
public int addition(int a,int b);
}
```

Applying Enterprise Beans 7-10

The Calculator interface is implemented by a stateless session bean CalculatorImplBean. Following code demonstrates the implementation of the methods of the Calculator interface:

```
package Test;
import javax.ejb.Stateless;
import javax.ejb.LocalBean;
@Stateless
@LocalBean
public class CalculatorImplBean implements Calculator{
@Override
public String sayHello(String name) {
return "Welcome to EJB"+" "+name;
}
@Override
public int addition(int a, int b) {
 return a+b;
 }
}
```

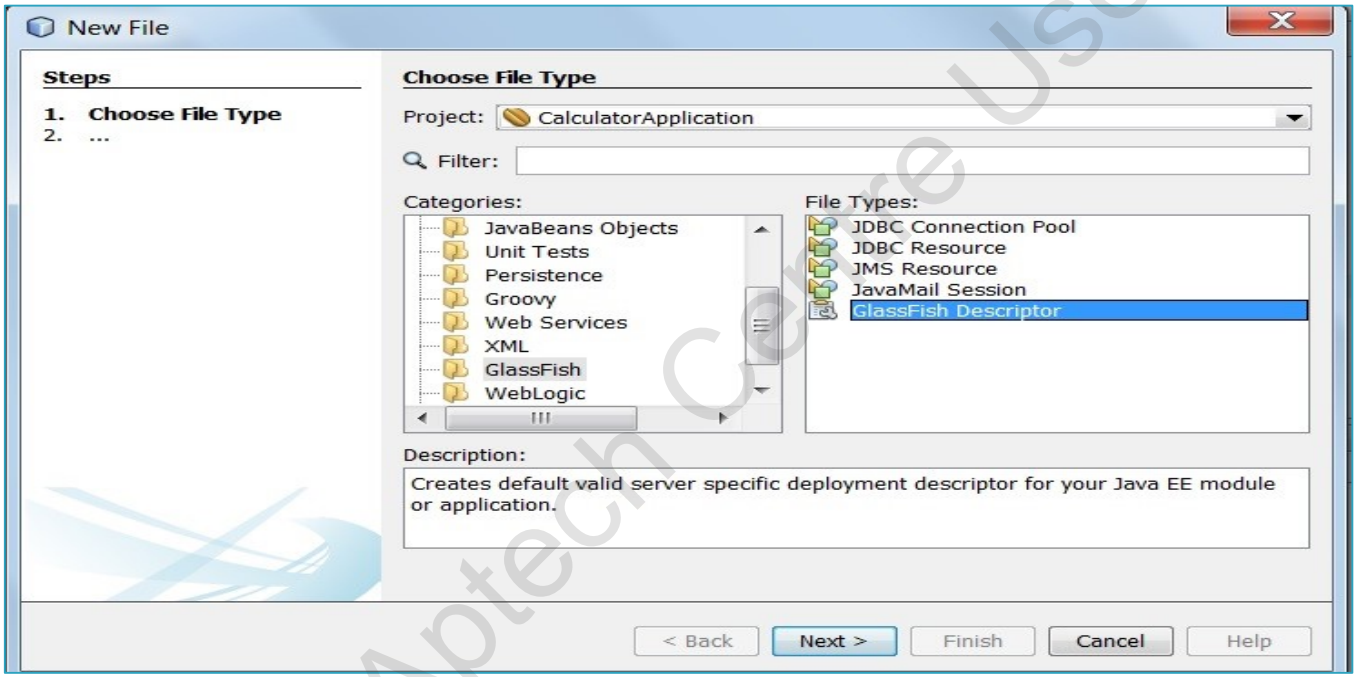
Applying Enterprise Beans 8-10

Following is the code of the client Main.java:

```
package Test;
import java.util.Properties;
import javax.naming.InitialContext;
import javax.naming.NamingException;
public class Main {
   public static void main(String[] args) throws NamingException {
        // TODO code application logic here
InitialContext c = new InitialContext();
Object l;
 l = c.lookup("caljndi”);
Calculator r=(Calculator)l;
System.out.println(r.sayHello(" Bruce "));
System.out.println( r.addition(5,10));
    }
}
```

Applying Enterprise Beans 9-10

The Enterprise bean should now be configured on the server, for this configuration create deployment descriptor as shown in the following figure:



There will always be only one glassfish descriptor in the package. The file is created with the name glassfish-ejb-jar.xml. Edit the XML file to specify the JNDI name.

Applying Enterprise Beans 10-10

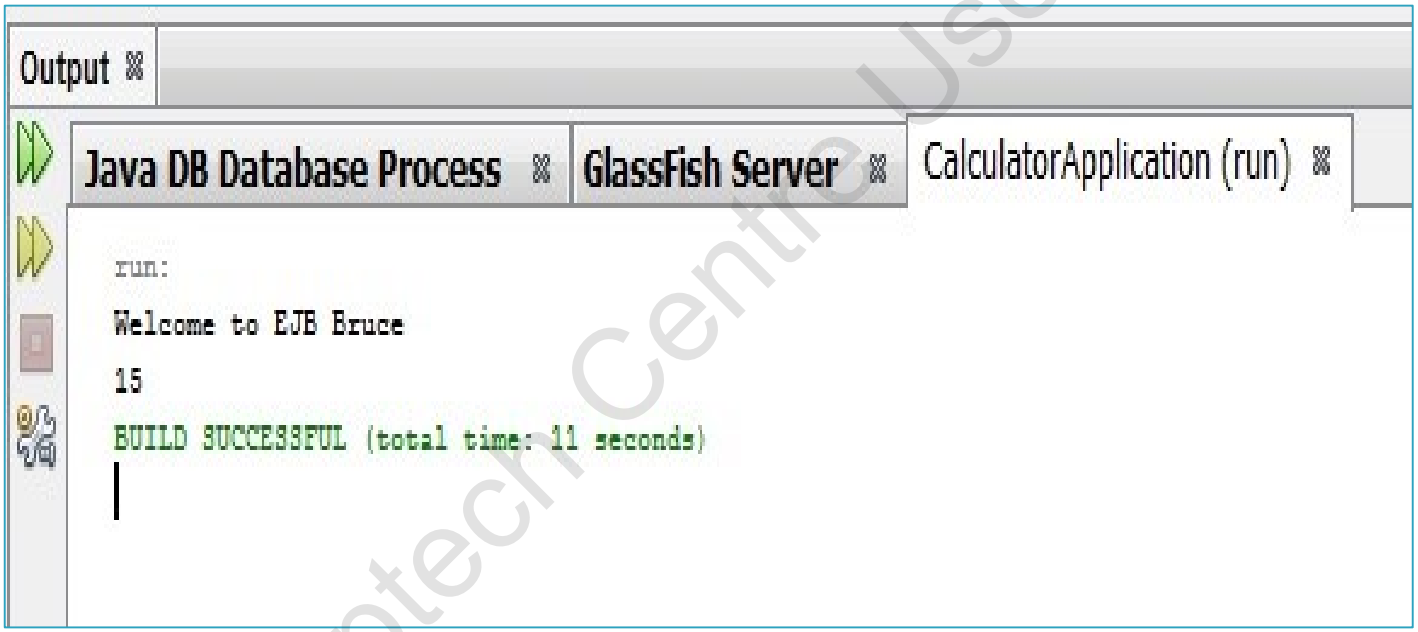
Following code demonstrates the GlassFish deployment descriptor:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE glassfish-ejb-jar PUBLIC "-//GlassFish.org//DTD
GlassFish Application Server 3.1 EJB 3.1//EN"
"http://glassfish.org/dtds/glassfish-ejb-jar_3_1-1.dtd">
<glassfish-ejb-jar>
  <enterprise-beans>
    <ejb>
      <ejb-name>CalculatorImplBean</ejb-name>
      <jndi-name>caljndi</jndi-name>
    </ejb>
  </enterprise-beans>
</glassfish-ejb-jar>
```

Set the JNDI name in the XML file to caljndi which is given as parameter to the lookup() method in the client file (Main.java). Clean and build the application once the GlassFish Server configuration is done.

Deploying the Code

Once the build is successful, deploy the code by right-clicking the package. Following figure shows the output of the application:



Creating a Web Client and External Application Client for an Enterprise Bean 1-28

An application client can access the bean methods through remote interface.

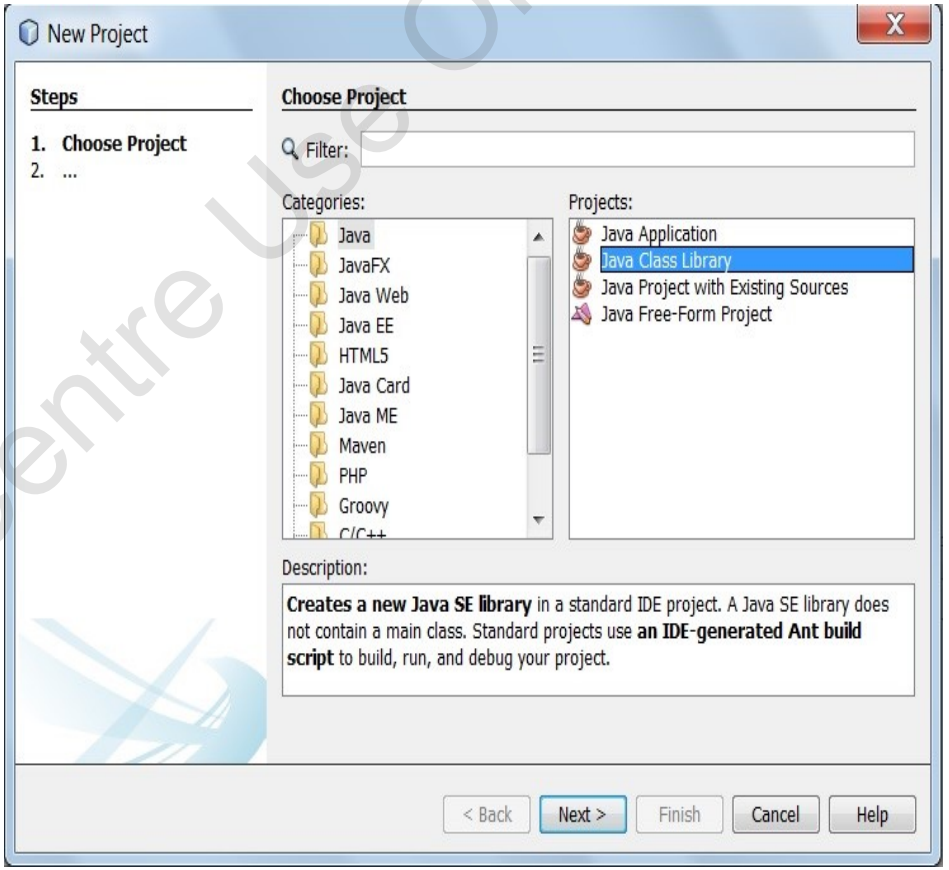
Following components are created to create the communication bean and an application client:

- The remote interface
- An enterprise application with a bean method. This bean method is created in a session bean so that it can be accessed by request
- Application client
- Web client

Creating a Web Client and External Application Client for an Enterprise Bean 2-28

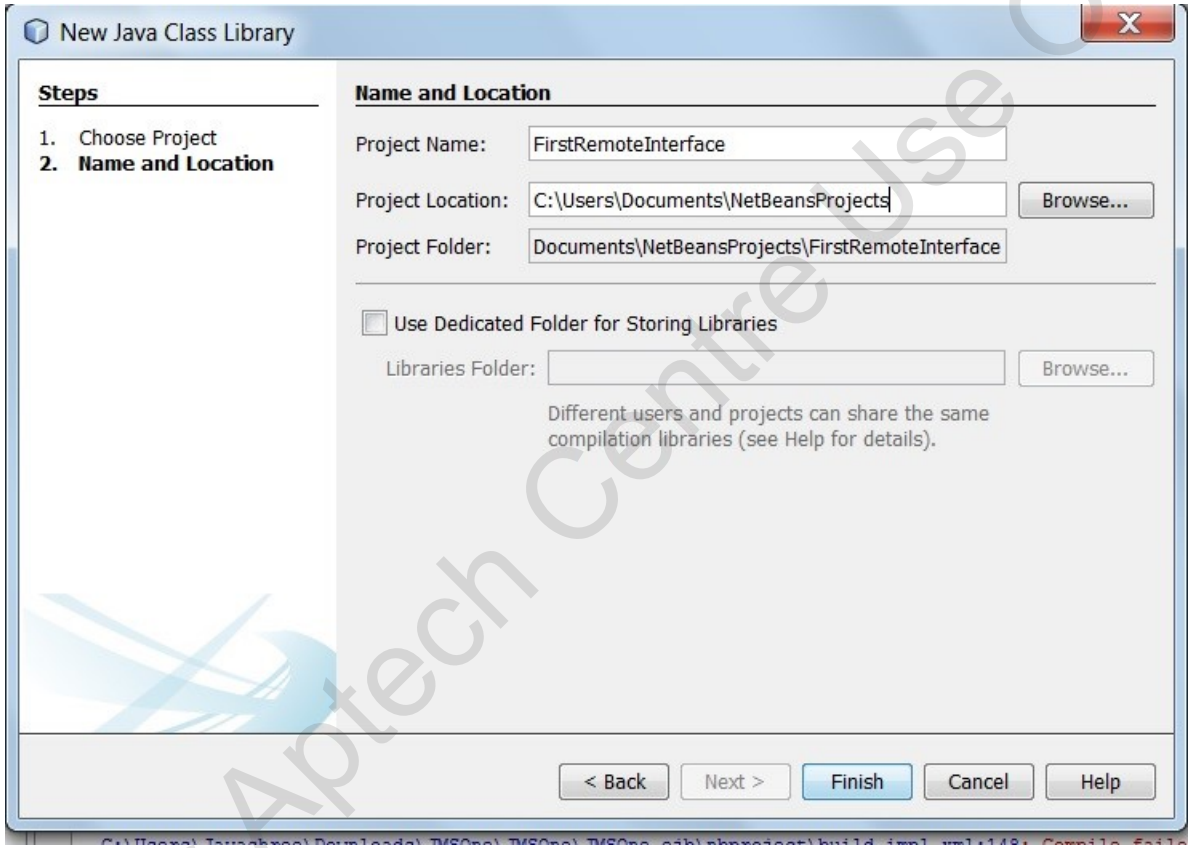
Creating a remote interface

- Create a new Java Class Library project in the NetBeans IDE by clicking File → New Project → Java → Java Class Library from the New Project dialog box as shown in the figure.



Creating a Web Client and External Application Client for an Enterprise Bean 3-28

Name the session interface as 'FirstRemoteInterface' as shown in the following figure:

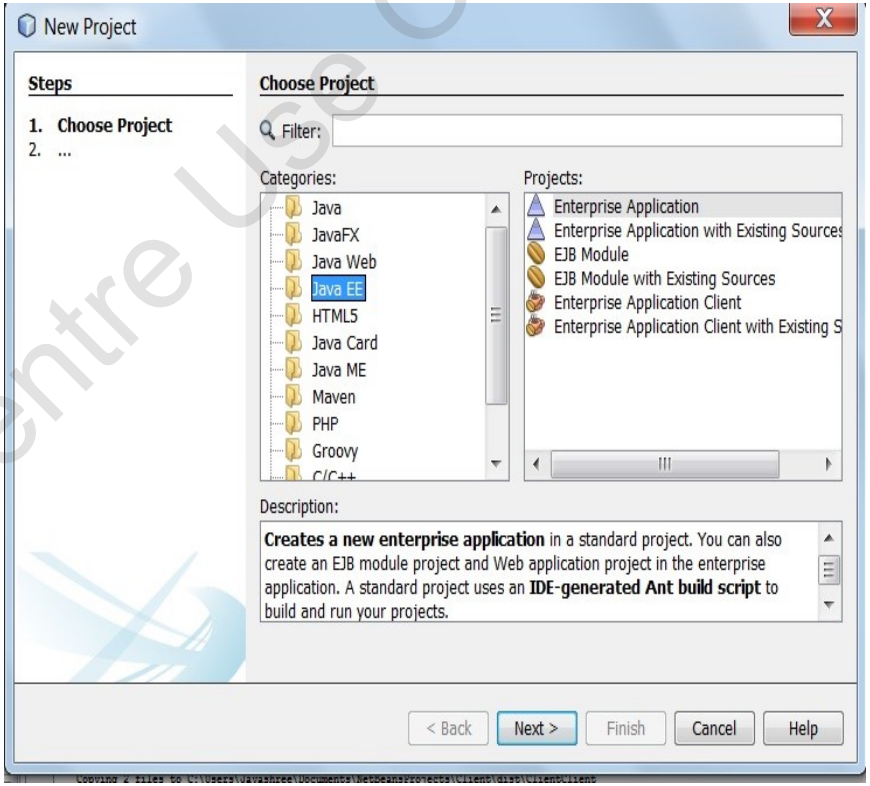


Once the remote interface is created, create an Enterprise Application which consists of the enterprise bean. Configure this enterprise bean to be accessed by the application client through the remote interface.

Creating a Web Client and External Application Client for an Enterprise Bean 4-28

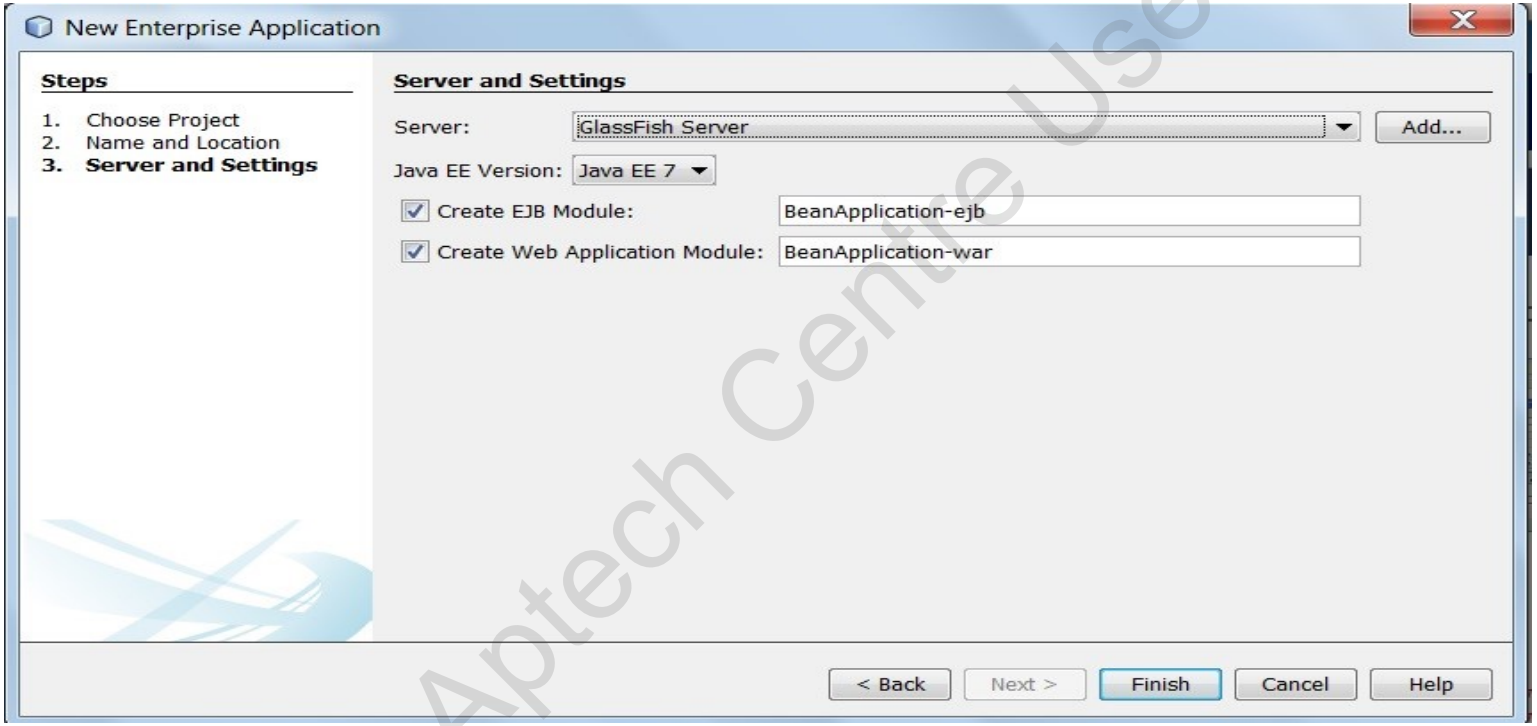
Creating the Enterprise application

- To create an enterprise application named 'BeanApplication' with EJB module, click File → New Project → Java EE → Enterprise Application as shown in the figure.
- Click Next.



Creating a Web Client and External Application Client for an Enterprise Bean 5-28

Specify the name of the application as BeanApplication and click Next.
Select the options from the Server and Settings screen as shown in the following figure:

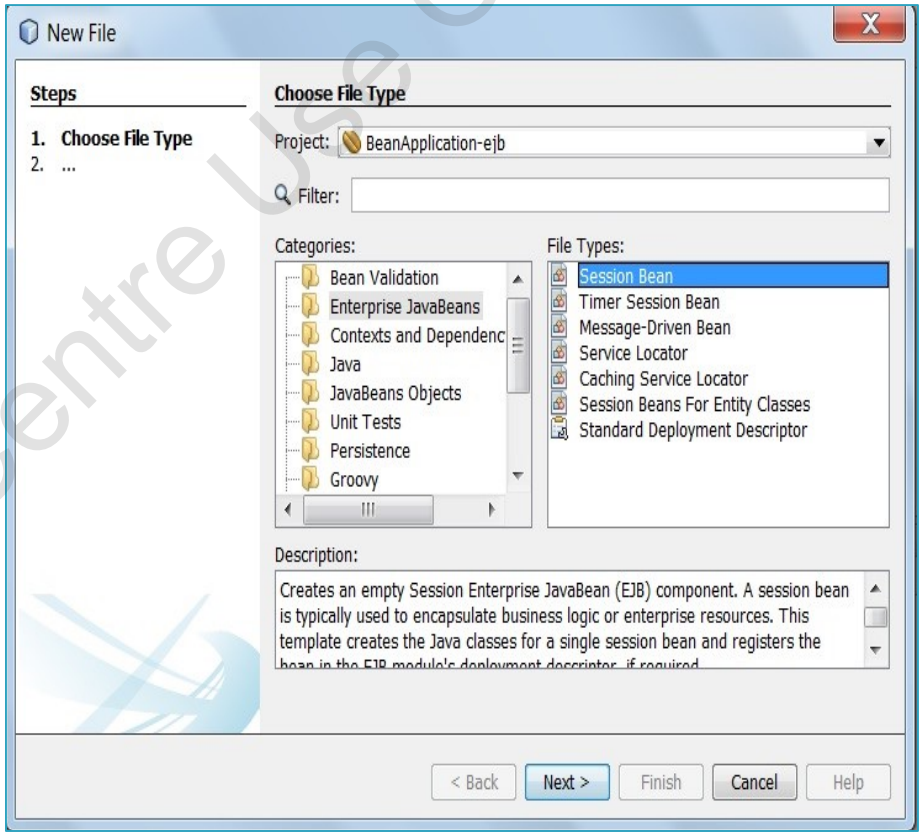


Ensure that both the Create EJB Module and Create Web Application Module checkboxes are selected. Click Finish.

Creating a Web Client and External Application Client for an Enterprise Bean 6-28

Creating the Session Bean

- To create a session bean in the EJB module, right-click the BeanApplication-ejb project and select New → Other → Enterprise JavaBeans → Session Bean as shown in the figure.
- Click Next.

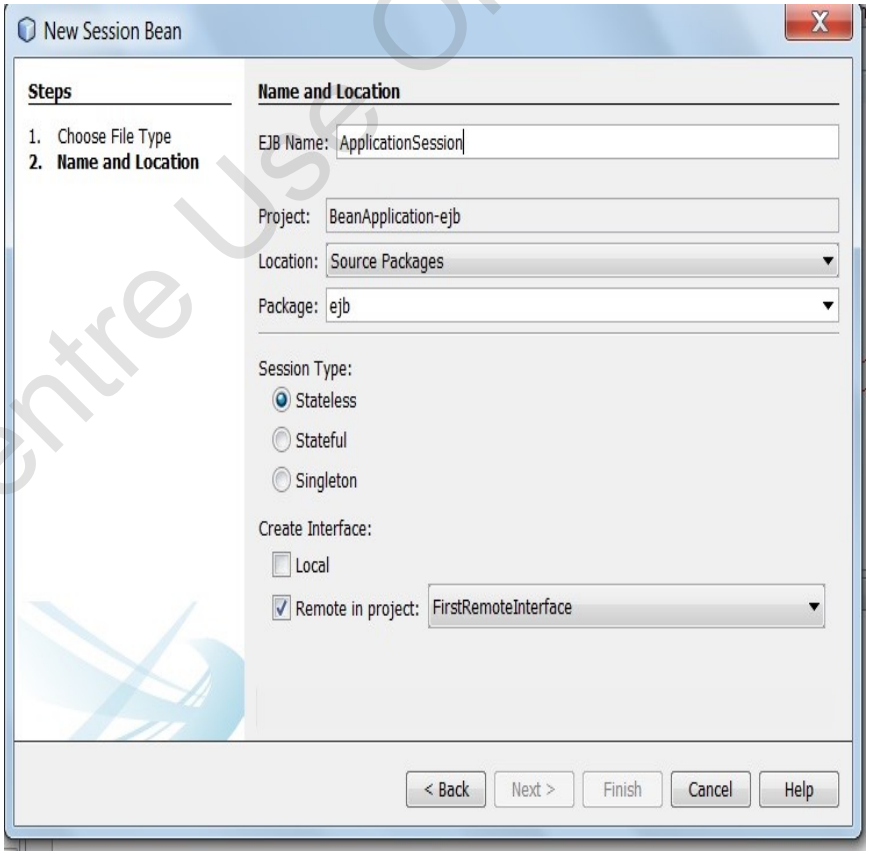


Creating a Web Client and External Application Client for an Enterprise Bean 7-28

Specify the name of the bean as ApplicationSession and package name as ejb.

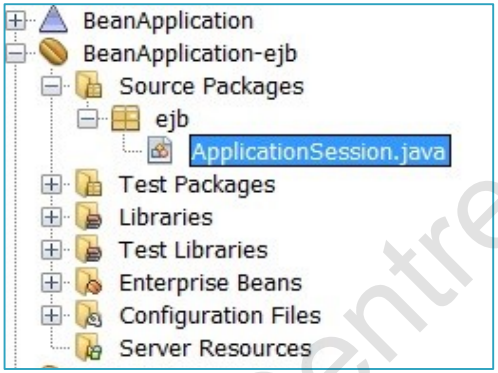
Ensure that Stateless option is selected.

Select the Remote in project checkbox. The FirstRemoteInterface created earlier will automatically get added to the drop-down.

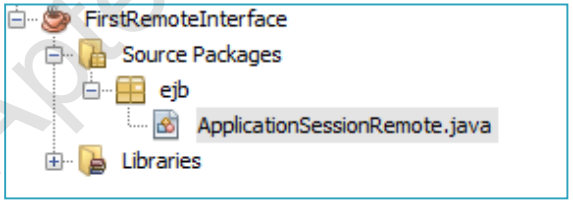


Creating a Web Client and External Application Client for an Enterprise Bean 8-28

Click Finish. The Session Bean is created within the ejb folder as shown in the following figure:



A corresponding remote interface named ApplicationSessionRemote is automatically created in the FirstRemoteInterface project as shown in the following figure:

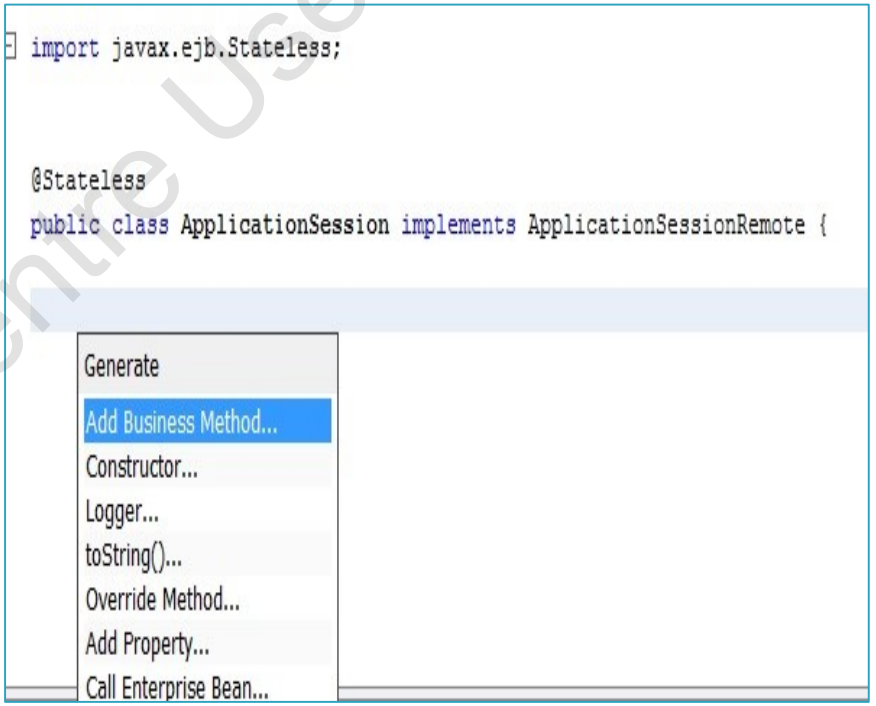


The ApplicationSession implements the remote interface defined earlier.

Creating a Web Client and External Application Client for an Enterprise Bean 9-28

Creating business methods

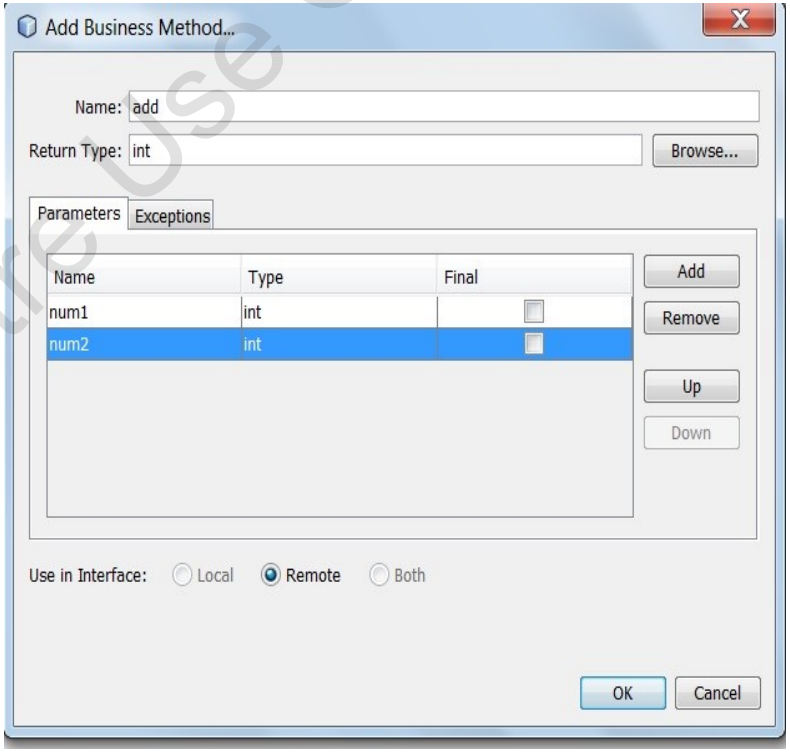
- Define business methods in the bean that can be accessed by the client, the IDE provides options for the code to be inserted directly by clicking Alt+Insert. It gives the options as shown in the figure.
- The menu can also be opened by right-clicking the editor and selecting Insert Code.



Creating a Web Client and External Application Client for an Enterprise Bean 10-28

Click Add Business Method. The Add Business Method dialog box is displayed.

Specify the method details in the dialog box as shown in the figure.



Creating a Web Client and External Application Client for an Enterprise Bean 11-28

Click OK. The method will be added to the ApplicationSession bean as shown in following figure:

```
@Stateless
public class ApplicationSession implements ApplicationSessionRemote {

    // Add business logic below. (Right-click in editor and choose
    // "Insert Code > Add Business Method")
    @Override
    public int add(int num1, int num2) {
        return 0;
    }


}
```

Simultaneously, the method signature is also added to the ApplicationSessionRemote file as shown in the following figure:

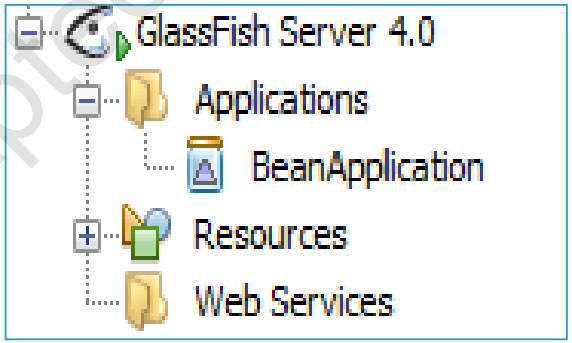
```
package ejb;

import javax.ejb.Remote;

@Remote
public interface ApplicationSessionRemote {

    int add(int num1, int num2);

}
```

Creating a Web Client and External Application Client for an Enterprise Bean 12-28

Following demonstrates how to modify the code in the ApplicationSession bean's add method:

```
public int add(int num1, int num2) {
        return num1+num2;
    }
```

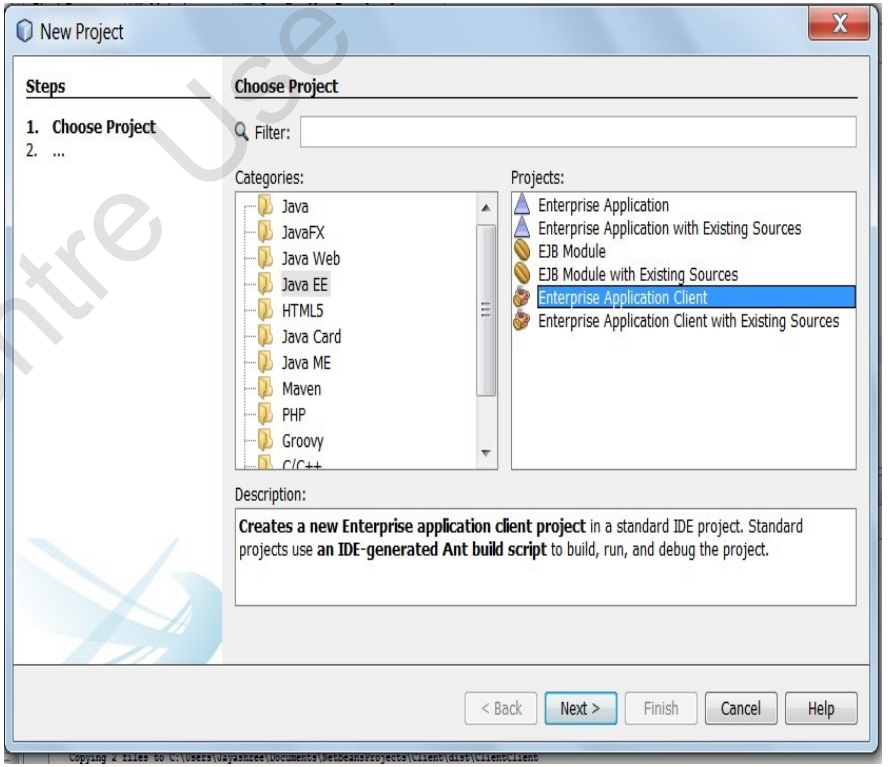
Build and deploy the enterprise application after defining the method in the session bean. Right-click the BeanApplication and select Deploy. Application can be viewed in the Services tab under the GlassFish Server as shown in the following figure:



Creating a Web Client and External Application Client for an Enterprise Bean 13-28

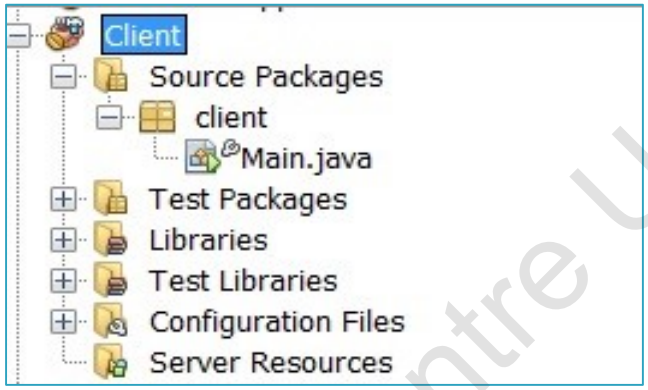
Creating the Application Client

- To create an external enterprise application client, click File → New Project → Java EE → Enterprise Application Client as shown in the figure.
- Click Next. Name the project as Client and click Next.
- Select the server as Glassfish and JavaEE7 as the Java EE Version and click Finish.

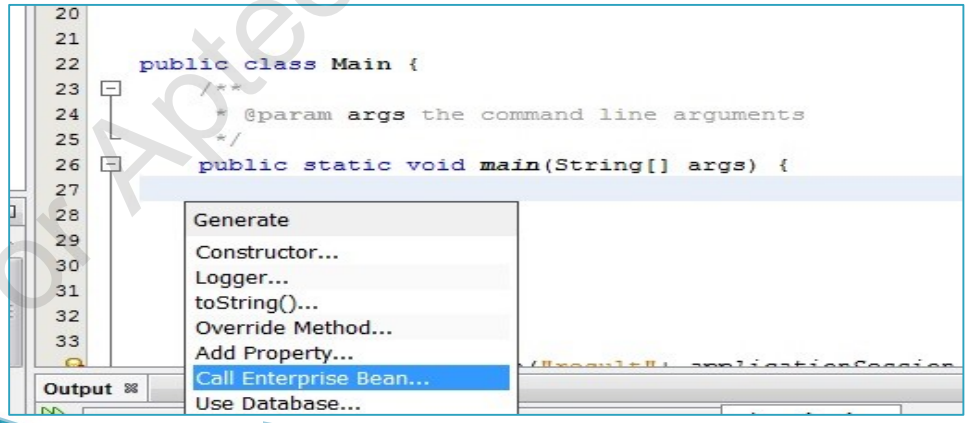


Creating a Web Client and External Application Client for an Enterprise Bean 14-28

The hierarchy of the Client application is shown in the following figure:



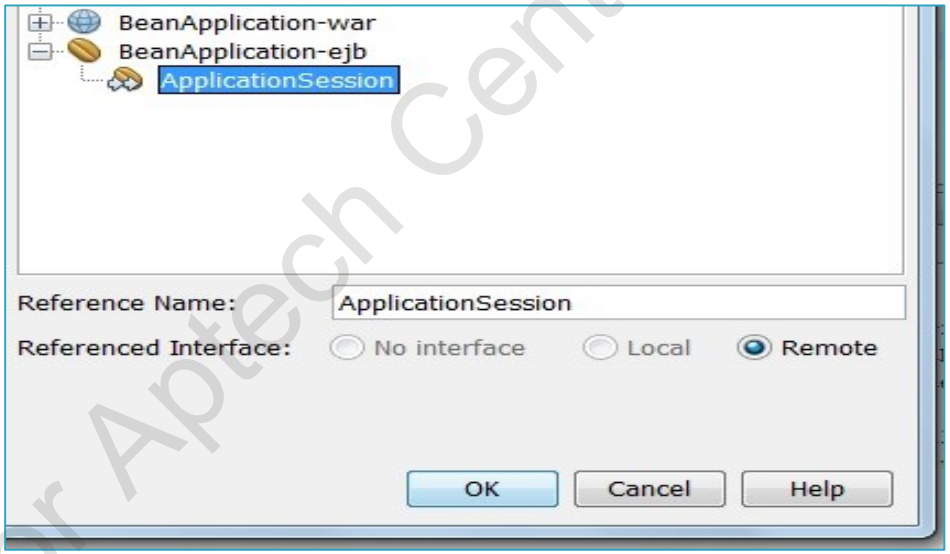
To invoke the ApplicationSession bean method from the client, you need to add a reference to the remote interface. To do this, press Alt+Insert and select 'Call Enterprise Bean' as shown in the following figure:



Creating a Web Client and External Application Client for an Enterprise Bean 15-28

On selecting the option 'Call Enterprise Bean', the IDE will pop a wizard in which the developer can choose the bean to be invoked.

Select the ApplicationSession bean as shown in the following figure:



Creating a Web Client and External Application Client for an Enterprise Bean 16-28

Add the reference of the remote interface to the main class of the client application as shown in the following figure:

```
public class Main {
    @EJB
    private static ApplicationSessionRemote applicationSession;

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        // TODO code application logic here


    }

}
```

After adding the reference, the developer can invoke the methods of the bean exposed by the remote interface.

Creating a Web Client and External Application Client for an Enterprise Bean 17-28

Following demonstrates the code used by the client application to invoke the add method of the ApplicationSession bean:

```
package client;
import ejb.ApplicationSessionRemote;
import javax.ejb.EJB;
public class Main {
    @EJB
    private static ApplicationSessionRemote
applicationSession;
    public static void main(String[] args) {

        System.out.println("Result: "+
applicationSession.add(2,4));
}    }
```

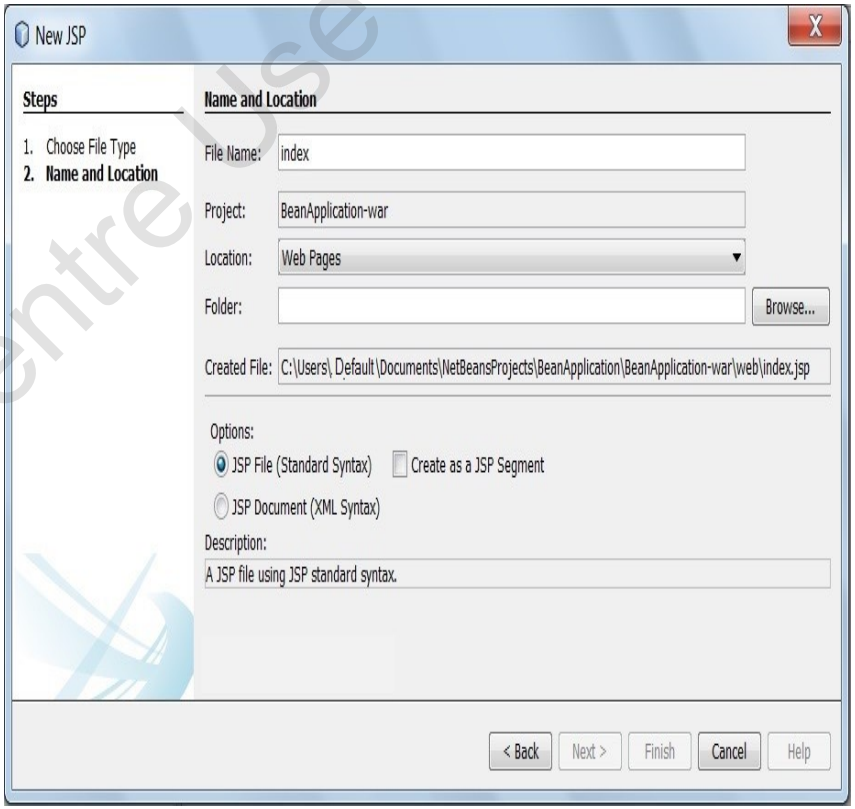
On deploying and running the application client, the output is shown in the following figure:

```
Result: 6
run:
BUILD SUCCESSFUL (total time: 18 seconds)
```

Creating a Web Client and External Application Client for an Enterprise Bean 18-28

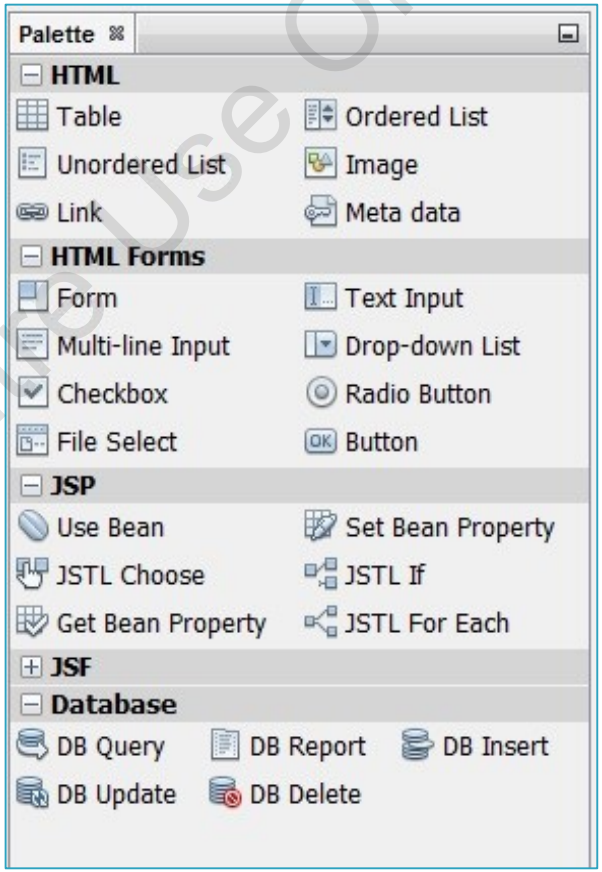
Creating Web client

- In the BeanApplication-war module, right-click the Web Pages folder and select New → JSP. The New JSP dialog box is displayed. Specify the name as index as shown in the figure:



Creating a Web Client and External Application Client for an Enterprise Bean 19-28

Add input components to the JSP page. In order to add input components activate the 'Palette' with a combination of Ctrl+Shift+8. This will invoke the palette as shown in the figure:



Creating a Web Client and External Application Client for an Enterprise Bean 20-28

Drag and drop the components from the palette into the body section of the JSP. All the UI components should be placed in a form component, therefore, first drop the form component in the JSP page.

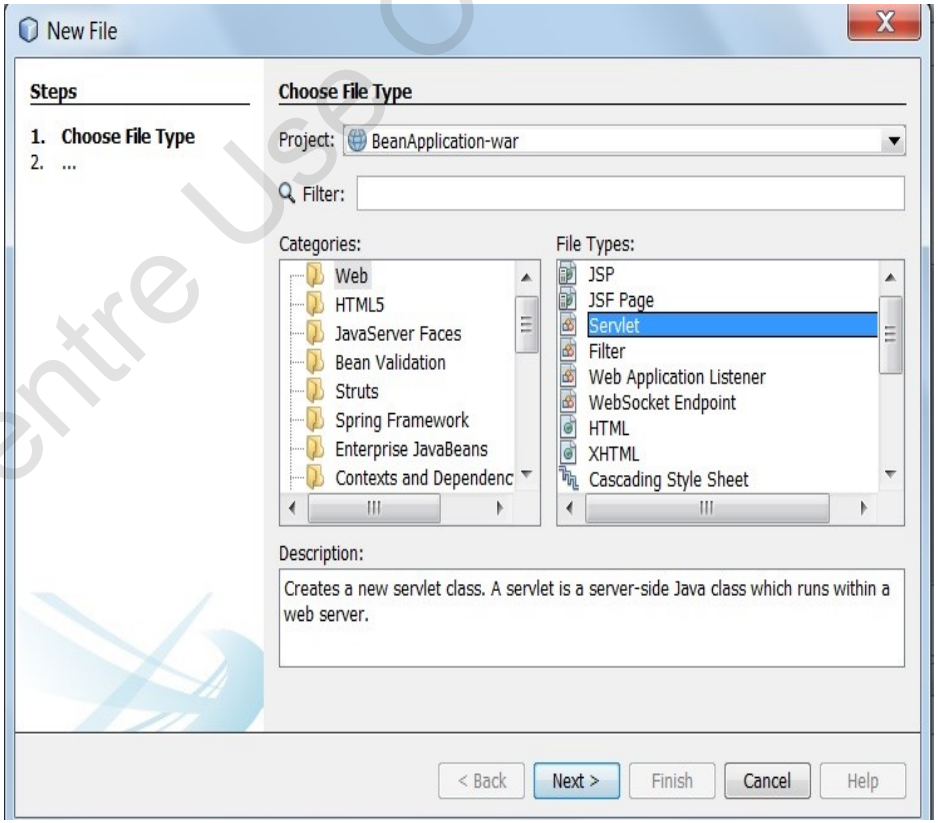
Given code demonstrates the code of the index.jsp file.

```
<%@page contentType="text/html"
pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
    <head>
        <meta http-equiv="Content-Type"
content="text/html; charset=UTF-8">
        <title>Web Client</title>
    </head>
    <body>
        <form name="Addition"
action="Addition" method ="GET">
            <input type="text"
name="num1" value="" size="5" />
            <input type="text"
name="num2" value="" size="5" />
            <input type="submit"
value="Add"  />
        </form>
    </body>
</html>
```

Creating a Web Client and External Application Client for an Enterprise Bean 21-28

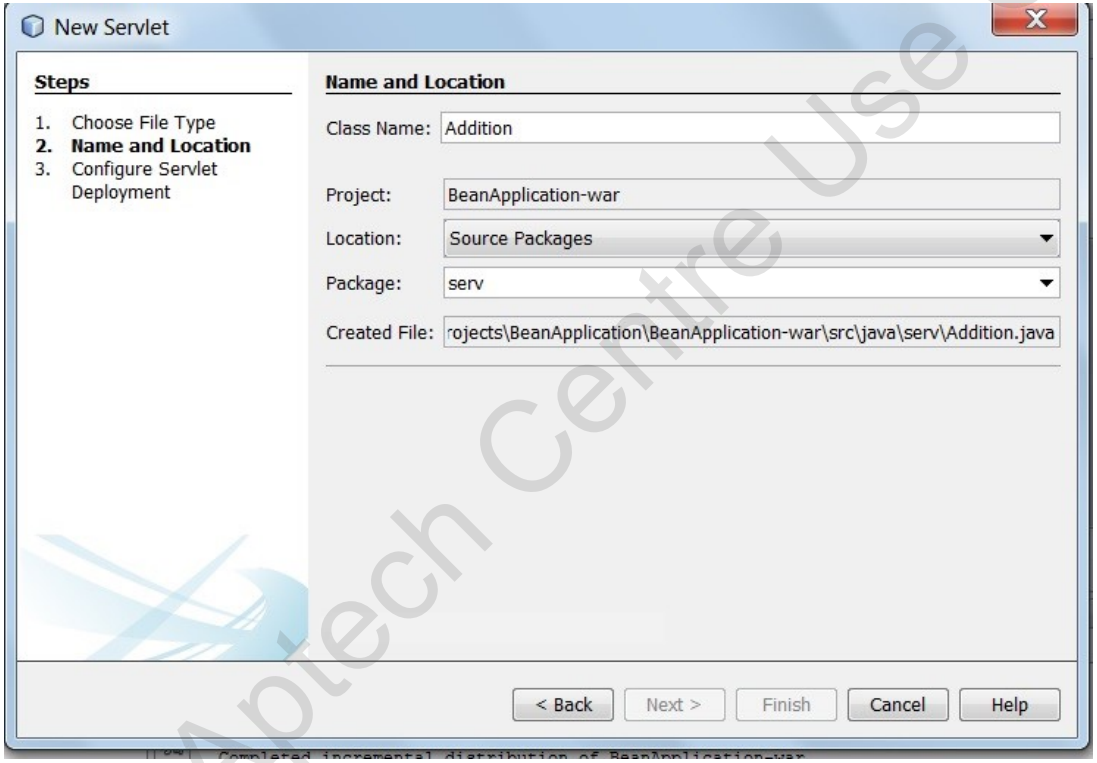
Creating the servlet for JSP

- The servlet will be used to write the business logic to invoke the ApplicationSession bean. In order to create the servlet, right-click the BeanApplication-war project and select New → Other → Web → Servlet as shown in the figure:



Creating a Web Client and External Application Client for an Enterprise Bean 22-28

Specify the package and name of the servlet as shown in the following figure:

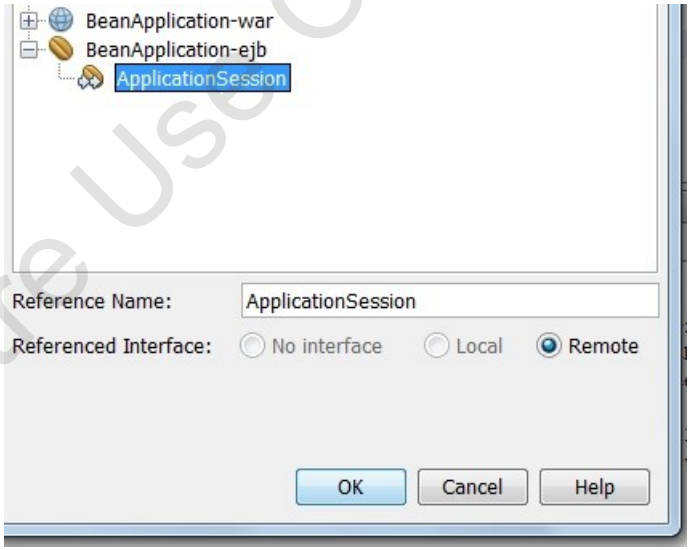


Click Next. On the next screen, select the 'Add information to deployment descriptor (web.xml) ' checkbox. Click Finish. The servlet named Addition will be created with the processRequest, doGet, and doPost methods.

Creating a Web Client and External Application Client for an Enterprise Bean 23-28

Now, add reference of the remote interface of the ApplicationSession bean to be invoked. Right-click in the editor and select Insert Code → Call Enterprise Bean.

Select the ApplicationSession bean as shown in the figure.



Click OK. The remote interface reference will be added to the servlet as follows:

```
@EJB
private ApplicationSessionRemote applicationSession;
```

Creating a Web Client and External Application Client for an Enterprise Bean 24-28

Add the following code in the try block of the processRequest() method:

```
int n1,n2;
n1 = Integer.parseInt(request.getParameter("num1"));
n2 = Integer.parseInt(request.getParameter("num2"));
```

- This piece of code converts the String parameters read from the JSP page to 'int' type.

Add the following code in the HTML block of the processRequest() method to invoke the addition operation:

```
out.println("<h2> Result: " + applicationSession.add(n1, n2)+
"</h2>" );
```

- These statements should be added in the body section of the HTML code generated from the servlet.

Creating a Web Client and External Application Client for an Enterprise Bean 25-28

Following code demonstrates the final code of the servlet:

```
package serv;
import ejb.ApplicationSessionRemote;
import java.io.IOException;
import java.io.PrintWriter;
import javax.ejb.EJB;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
public class Addition extends HttpServlet {
   @EJB
    private ApplicationSessionRemote applicationSession;
    protected void processRequest(HttpServletRequest request,
HttpServletResponse response)
            throws ServletException, IOException {
        response.setContentType("text/html;charset=UTF-8");
        try (PrintWriter out = response.getWriter()) {
            int n1,n2;
            n1 =
```

Creating a Web Client and External Application Client for an Enterprise Bean 26-28

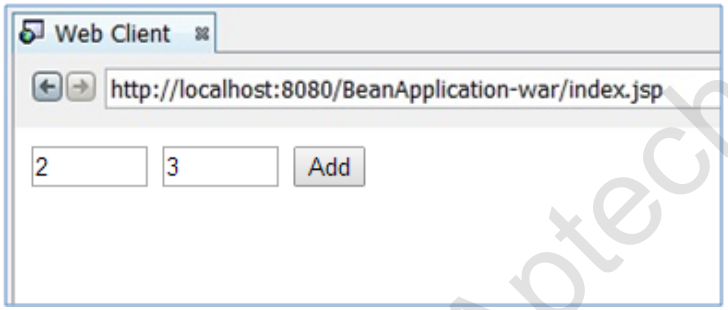
```
Integer.parseInt(request.getParameter("num1"));
            n2 = Integer.parseInt(request.getParameter("num2"));
            /* TODO output your page here. You may use following sample
code. */
            out.println("<!DOCTYPE html>");
            out.println("<html>");
            out.println("<head>");
            out.println("<title>Servlet Addition</title>");
            out.println("</head>");
            out.println("<body>");
            out.println("<h2> Result: " + applicationSession.add(n1,
n2)+ "</h2>" );
            out.println("</body>");
            out.println("</html>");
        }
    }
```

Creating a Web Client and External Application Client for an Enterprise Bean 27-28

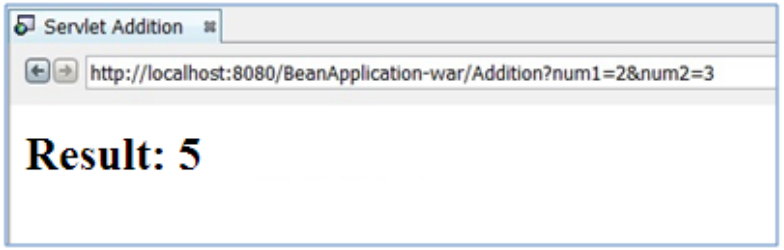
```
 @Override
    protected void doGet(HttpServletRequest request,
HttpServletResponse response)
            throws ServletException, IOException {
        processRequest(request, response);
    }
    @Override
    protected void doPost(HttpServletRequest request,
HttpServletResponse response)
            throws ServletException, IOException {
        processRequest(request, response);
    }
    @Override
    public String getServletInfo() {
        return "Short description";
    }// </editor-fold>
}
```

Creating a Web Client and External Application Client for an Enterprise Bean 28-28

Deploy the BeanApplication enterprise application. Right-click the BeanApplication enterprise application and select Run. The index.jsp page is displayed. Specify 2 and 3 in the textboxes as shown in the following figure:



Click Add. The resultant page generated by the servlet which displays the result is shown in the following figure:



Summary

- Enterprise beans can be session beans and message driven beans.
- There are three variants of session beans – stateful, stateless, and singleton session beans.
- There are three types of clients which access beans – local clients, remote clients, and Web services. These clients can access the beans through business interface and no-interface views.
- Each session bean goes through various states during its life cycle. These states are different for stateless, stateful, and singleton beans.
- Stateful sessions can be passive but stateless, singleton, and message driven beans cannot be passive.