Developing Java Web Services

Session: 5

JAX-WS

Objectives

- Explain Web services on Java Enterprise Edition platform
- Explain Java API for XML based Web services (JAX-WS)
- Explain the need of JAX-WS
- Explain the features and standards of JAX-WS
- Explain the JAX-WS architecture
- Explain JAX-WS annotations
- Explain JAX-WS programming model

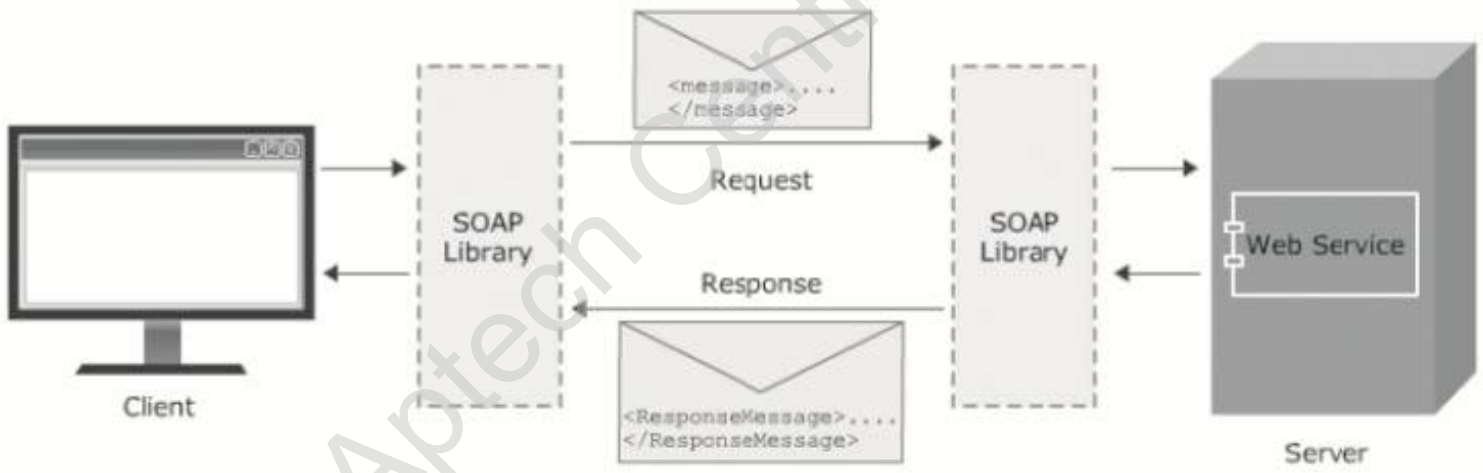
Enhancements of Web Services on Java EE Platform

- Today, there is a need to develop enterprise applications with fewer resources in lesser time and effort.
- One of the platforms used to develop enterprise applications and Web services is Java Enterprise Edition (Java EE).
- It has more annotations, lesser XML configurations, more Plain Old Java Object (POJO), and simple packaging.
- Following table shows some of the new technologies included on the Java EE platform:

Technology	Description
Java API for XML based Web Services (JAX-WS)	It is a Java API, to create Web services on a Java platform.
Java API for RESTful Web Services (JAX-RS)	It is a Java API to support creation of Web services based on REpresentational State Transfer (REST) architecture.
Dependency Injection	It is a technique that provides the required objects to the software components.

‘Big’ Web Services

- ‘Big’ Web services or ’SOAP-based’ services are based on JAX-WS in Java EE.
- SOAP is a standard protocol that defines the architecture and message formats in XML language.
- It follows HTTP protocol for request-and-response model on the Web.
- Following figure shows SOAP-based communication:



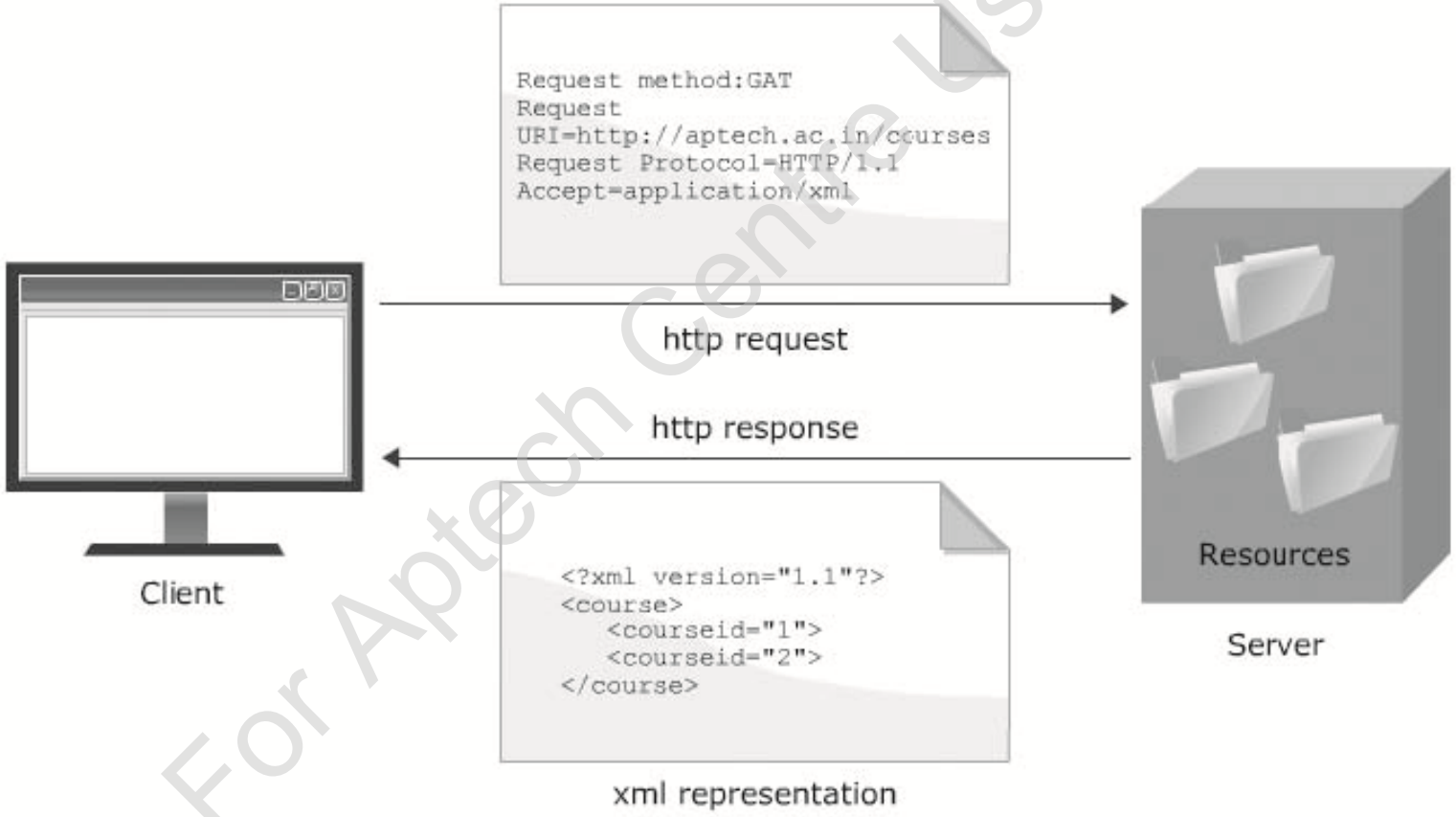
RESTful Web Services 1-2

- Is a Web application that is based on client-server architecture called as the REST architecture.
- Has a request-and-response model.
- Uses HTTP, which is based on Uniform Resource Identifier (URI) on the hyperlink, to access resources.
- Following table shows the mappings of HTTP methods to their CRUD actions:

HTTP Method	CRUD Action
POST	Create a new resource from the request data
GET	Retrieve a resource
PUT	Update a resource from the request data
DELETE	Delete a resource

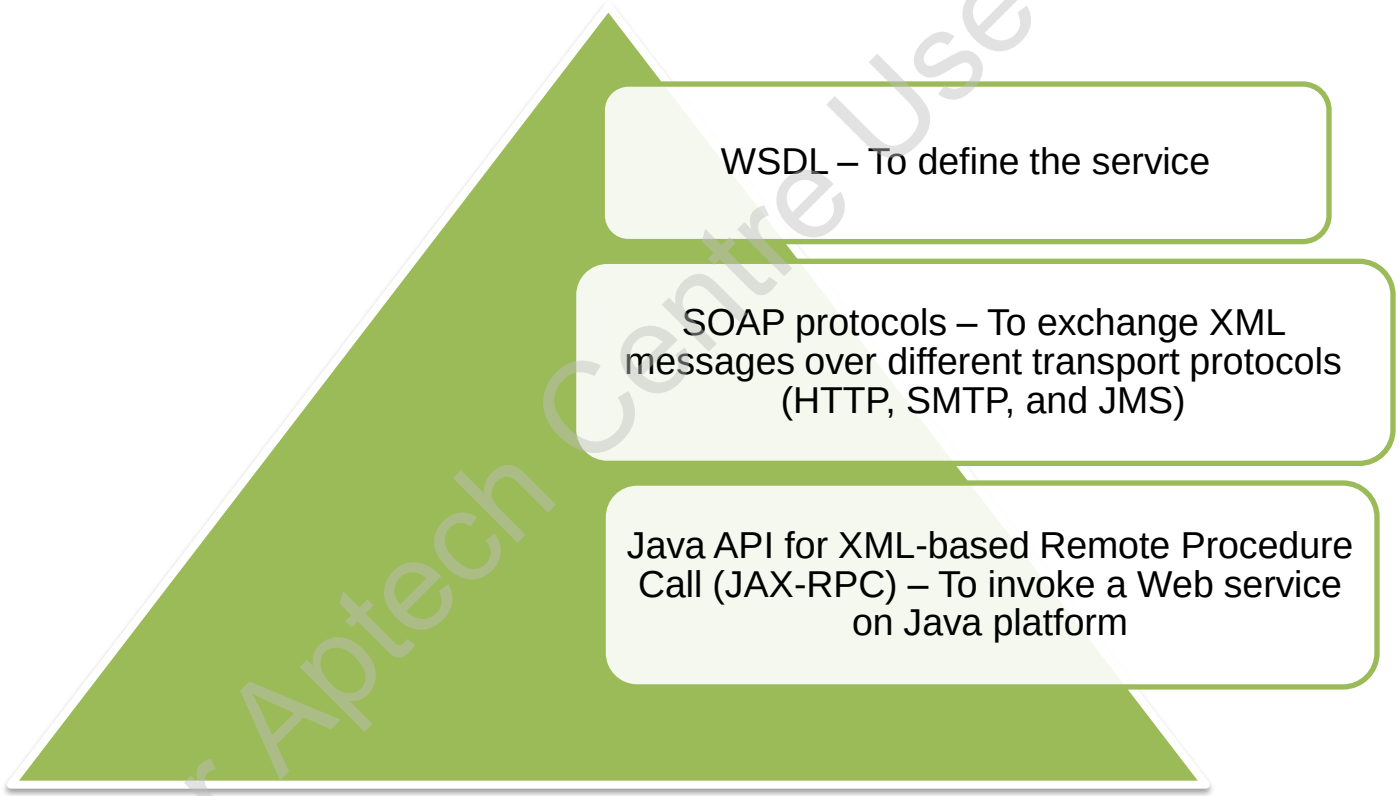
RESTful Web Services 2-2

- In REST-based communication, a client initiates the request for the resource. In response, the client receives representation of the resource.
- Following figure shows REST-based communication:



Java API for XML Web Services (JAX-WS)

- Web services support a few standards that help to develop interoperable applications. The key standards are as follows:

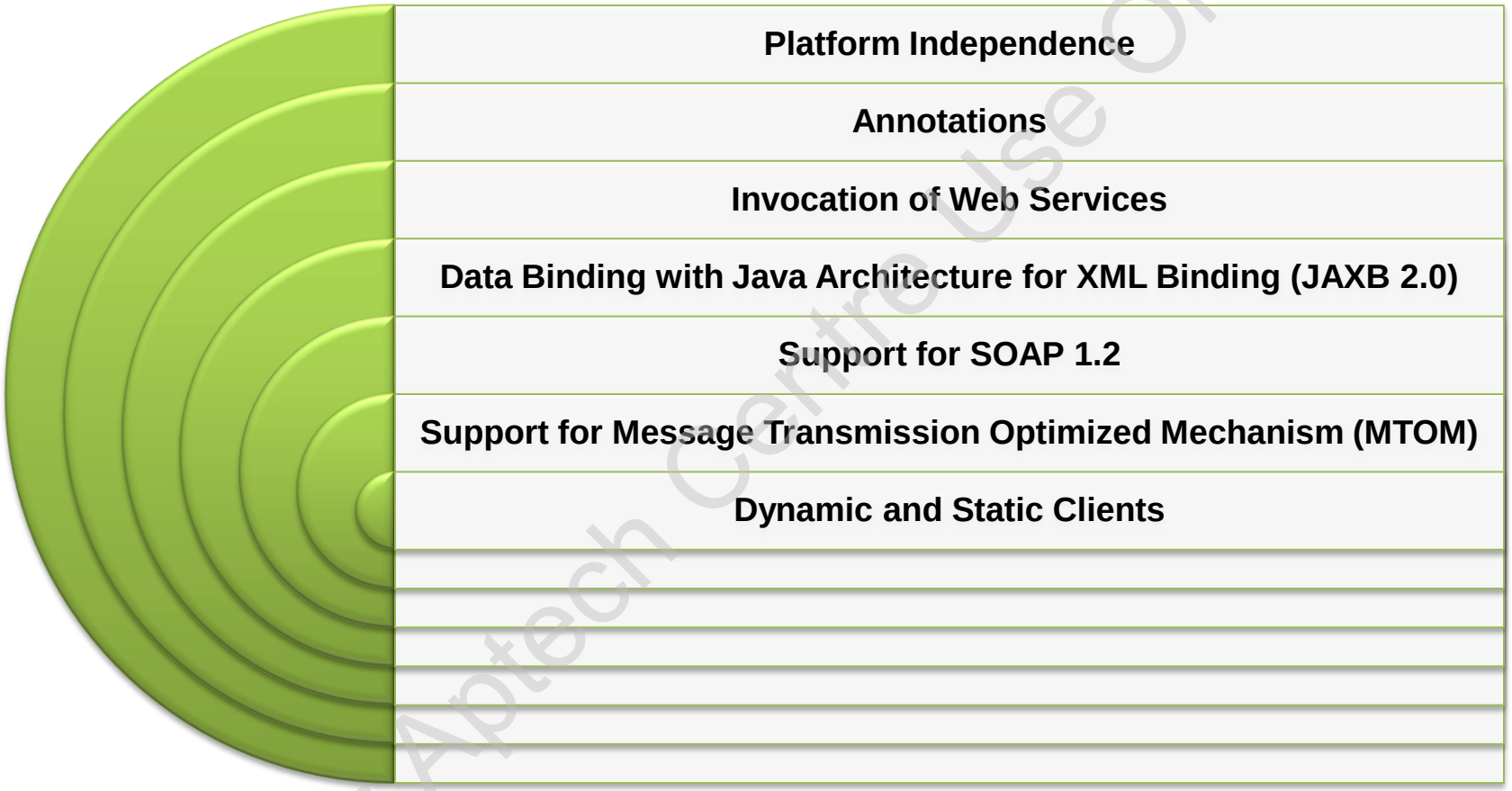


JAX-WS Standards

- Following table shows the new standards of JAX-WS programming model:

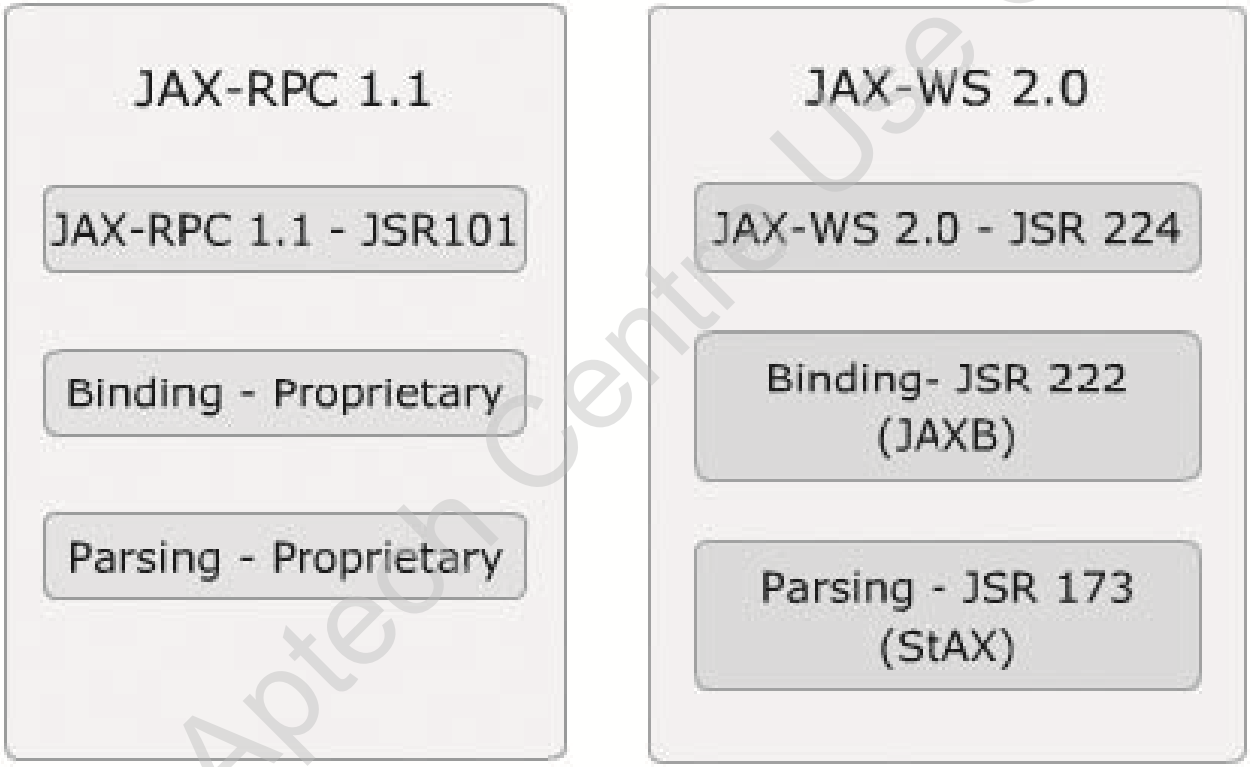
Standard	Description
SOAP 1.2	It is a lightweight protocol for structured information exchange of text and binary data.
XML/HTTP	It assists in transforming XML messages over transport protocol, HTTP without SOAP.
Web Services Interoperability (WS-I)	It has WS-I Basic Profile document that has specifications for SOAP and WSDL. WS-I Basic Profile 2.0 has the specified encoding styles, proxy generations, and dynamic invocation of interfaces.
Data Mapping Model	It specifies the mapping of XML elements to Java classes. JAXB 2.0 promotes the mappings for all XML schemas. It is supported by JAX-WS for data mapping.
Interface Mapping Model	It is used to map service interfaces with service implementation classes.
Dynamic Programming Model	It is used for message-oriented invocations and asynchronous invocations.
Handler Model	It processes the SOAP message before and after the messages are sent over the network in the Web service development.

Features of JAX-WS



Web Service Stack

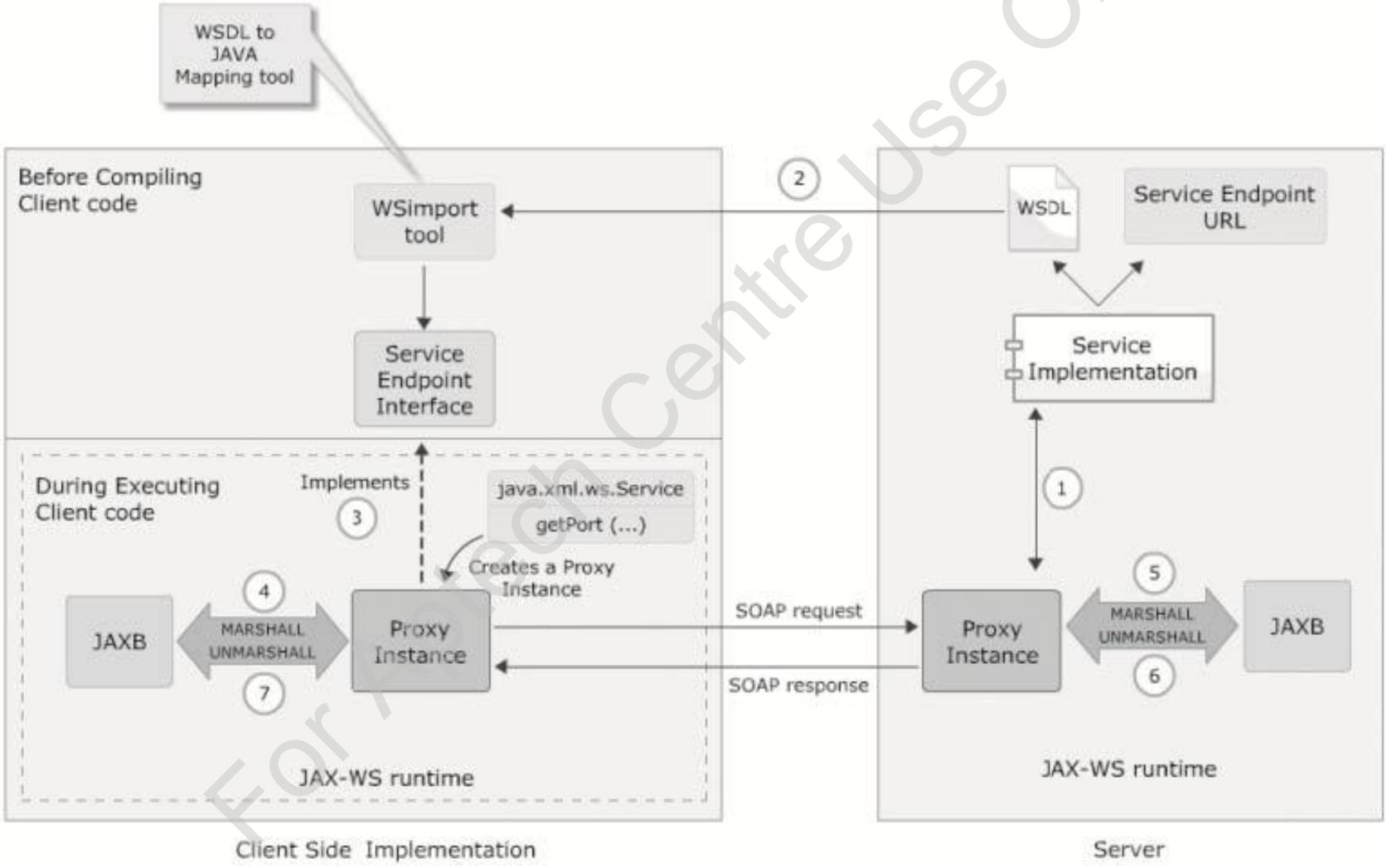
- Following figure shows the Web service stack for JAX-RPC and JAX-WS proposed by Sun:



- The figure shows the standards of JAX-WS and its comparison with JAX-RPC, which is the initial implementation of Web services.

JAX-WS Architecture 1-3

- Following figure shows JAX-WS architecture:



JAX-WS Architecture 2-3

- Following are the steps to invoke the Web:
  1. **At the server-side**: The development of JAX-WS starts with an annotated Web service implementation class. This implementation class is deployed on an application server. Then, the wsgen tool generates the WSDL file, which has information on service methods and the SEI. While deploying, a server-side proxy instance is generated to handle the request and response.
  2. **At the client-side**: The Web service is invoked using a command tool called wsimport. This tool is a WSDL to Java mapping tool. It generates the SEI on the client-side. Then, the client code for accessing the Web service is compiled and executed.
  3. **After the execution of client application code**: JAX-WS runtime generates a proxy class instance. This instance internally implements the generated SEI. The proxy class is invoked by the getPort() method from javax.xml.ws.Service class. The generated proxy instance is used for invoking the Web service method.
  4. **After invoking the method on Web service by the client**: The JAX-WS runtime uses JAXB to marshal the parameters of the method. Then, these objects are encapsulated in SOAP message in XML format. These are sent as a SOAP request across the network. The parameters are passed to proxy instance to be passed to the service present at the server.

JAX-WS Architecture 3-3

5. **After the SOAP request is received by the server**: The JAX-WS runtime environment uses JAXB to unmarshal the SOAP request. Unmarshalling converts the SOAP message having XML schema elements into Java objects. The message is given to the server-side proxy to execute the Web service method.
6. **At the server-side**: The result of the Web service method is marshalled again using JAXB. JAXB wraps the response in the SOAP message, to enable the server-side proxy to send it back to the client.
7. **At the client-side**: The JAX-WS runtime unmarshalls the SOAP response message using JAXB. Then, it passes the unmarshalled Java object to the proxy. The proxy returns the result from the Java object to the client application.

JAX-WS Annotations

- JAX-WS has a library of annotations to simplify Web services development.
- Annotations enable easy conversion of a POJO class to a Web service.
- At the server, annotations describe the way of accessing an implementation class as a Web service.
- At the client, annotations describe the way a client-side Java class accesses Web service.
- The application uses the metadata information defined on service implementation class or service interface. Specialized tools process the metadata to generate the underlying code.
- Annotations provide attributes that can be used to keep the required information in the implementation class itself.
- Therefore, it is optional to have webservices.xml deployment descriptor file.

Web Services Metadata Annotations 1-5

@WebService Annotation

- Following table shows the attributes of javax.jws.WebService annotation:

Attribute	Description
name	It specifies the name of the Web service. It is represented by wsdl:portType. Its default value is the name of Java class/interface.
targetNamespace	It specifies the XML namespace of the XML elements. Its default value is the namespace mapped to a package name of Web service.
serviceName	It specifies the service name of the Web service. It is mapped in the WSDL file using wsdl:service. Its default value is the Java class/interface and service.
endpointInterface	It helps to separate the service interface from the implementation class. If this property is not specified, then the Web service interface is generated by the implementation class.
wsdlLocation	It indicates the relative/absolute Web address of the WSDL file that defines the Web service.

Web Services Metadata Annotations 2-5

@WebMethod Annotation

- Following table shows the attributes of javax.jws.WebMethod annotation:

Attribute	Description
operationName	It maps the method to wsdl:operation element in the WSDL file. It defines the operation type performed by the Web service method. The default value is the name of the Java method as a string.
action	It defines the action for this operation. It finds out SOAP action header value for SOAP bindings. Its default value is a string.

@WebResult Annotation

- Following table lists the attributes of javax.jws.WebResult annotation:

Attribute	Description
name	It specifies the variable name that is returned after invoking the Web service method. In document style operations, the name parameter is the local name of XML element, which is the return value.
targetNamespace	It specifies the XML namespace for the value returned. When the return value maps to an XML element, this is used to bind documents. The default value of targetNamespace is empty namespace, represented as “”.

Web Services Metadata Annotations 3-5

@WebParam Annotation

- Following table shows the attributes of javax.jws.WebParam annotation:

Attribute	Description
name	It specifies the name of the parameter.
targetNamespace	It specifies the XML namespace for the parameter. This attribute has to be applied only when the operation is document-style or the parameter maps to a header. Its default value is the Web service's namespace.
mode	It shows the direction in which the parameters are passed and returned in the method. Its values can be IN, OUT, and INOUT. Its default value is IN.

Web Services Metadata Annotations 4-5

@SOAPBinding Annotation

- Following table lists the attributes of javax.jws.soap.SOAPBinding annotation:

Attribute	Description
style	It defines the message encoding style of Web services. Its value can either be DOCUMENT or RPC. Its default value is javax.jws.soap.SOAPBinding.Style.DOCUMENT.
use	It defines the message formatting style in Web services. Its default value is LITERAL (javax.jws.soap.SOAPBinding.Use.LITERAL).
parameterStyle	It determines the way the method parameters are placed in the message body. Its default value can be BARE or WRAPPED. If the value is set to BARE, it implies that each parameter is placed as a child element in the message body. If the value is set to WRAPPED, it means that all input/output parameters are in a single element in the respective request/response message. Its default value is javax.jws.soap.SOAPBinding.ParameterStyle.WRAPPED.

Web Services Metadata Annotations 5-5

@SOAPMessageHandler Annotation

- Following table lists the attributes of javax.jws.soap.SOAPMessageHandler annotation:

Attribute	Description
name	Specifies the name of SOAP message handler. The default value of this attribute is the name of the class that implements the Handler interface.
className	Specifies name of the Handler class.
initParams	Specifies the array of the name/value pairs that will be passed to the handler during initialization.
roles	Specifies the list of SOAP roles that the handler implements.
headers	Specifies the list of headers that the handler processes.

@initParams Annotation

- Following table lists the attributes of javax.jws.soap.initParams annotation:

Attribute	Description
name	Specifies the name of the initialization parameter.
value	Specifies the value of the initialization parameter.

Core JAX-WS API Annotations 1-4

There are a few core annotations of JAX-WS API that specify the metadata associated with Web service implementations.

At runtime, these annotations simplify the process of developing Web services. They help to map Java to WSDL and schema.

They also respond to Web service invocations and help to control the JAX-WS runtime processes.

Core JAX-WS API Annotations 2-4

@RequestWrapper Annotation

- Following table lists the attributes of javax.xml.ws.RequestWrapper annotation:

Attribute	Description
localName	It describes the local name of the wrapper element in the message request. It is in the form of XML. Its default value is the name of the method annotated with @WebMethod.
targetNamespace	It indicates the namespace of the request wrapper element. Its default value is the targetNamespace of the SEI.
className	It indicates the name of the request wrapper class.

@ResponseWrapper Annotation

- Following table shows the attributes of javax.xml.ws.ResponseWrapper annotation:

Attribute	Description
localName	It describes the wrapper element's local name in the XML message response. Its default value is the method name annotated with @WebMethod.
targetNamespace	It indicates the namespace of the request wrapper element. Its default value is the targetNamespace attribute value of the SEI.
className	It indicates the name of the response wrapper class.

Core JAX-WS API Annotations 3-4

@WebServiceProvider Annotation

- Following table lists the attributes of javax.xml.ws.WebServiceProvider annotation:

Attribute	Description
wsdlLocation	It is a mandatory attribute. It defines the Web location of the WSDL file that describes the Web service.
portName	It defines the service end point and maps to the wsdl:port element of WSDL file.
serviceName	It defines the service endpoint and maps to the wsdl:service element of WSDL file. Its default value is the unqualified name of the Java class, which is appended with the service keyword.
targetNamespace	It specifies the WSDL generated from Web service and the XML namespace of the XML elements. Its default value is the namespace, which is mapped to a package name of the Web service.

Core JAX-WS API Annotations 4-4

@ServiceMode Annotation

- This annotation is used to develop JAX-WS Web services implementation class with the help of the Web service provider.

@WebServiceRef Annotation

- Following table lists the attributes of javax.xml.ws.WebServiceRef annotation:

Attribute	Description
name	It specifies the JNDI name of the resource. It maps the annotated method to its related Java bean property name. It maps to the default field name in case of field annotation.
type	It specifies the resource of Java type. It maps the annotated method type to its related Java bean property type. It maps to the default field name in case of annotated field.
mappedName	It specifies that a resource is mapped to a product specific name.
value	It specifies that javax.xml.ws.Service is extended by the service class. Its reference is of SEI.

Design Requirements of a JAX-WS Application

- The two types of service endpoint implementations supported by JAX-WS, which make services to work at XML message level are as follows:

The standard JavaBeans service endpoint interface

A new Provider interface

Requirements of Service Endpoints in JAX-WS

- Following table lists the requirements for JAX-WS service endpoints as Dos and Don'ts:

Dos	Don'ts
1. Use javax.jws.WebService or javax.jws.WebServiceProvider annotation to annotate the Web service implementation class. 2. Declare the business methods as public. 3. Use javax.jws.WebMethod annotation to declare the business methods. 4. javax.jws.WebMethod annotation should have JAXB compatible parameters and return types. This has to be done for business methods that are exposed to Web service clients. 5. The life cycle event callback methods can be contained in implementing class. The methods include javax.annotation.PostConstruct or javax.annotation.PreDestroy.	1. Do not mention the endpoint interface element explicitly in @WebService annotation as the SEI is implicitly defined. An implementation class can reference the SEI. 2. Do not declare business methods as static or final in the implementation class. 3. Do not declare the implementing class as abstract or final. 4. Do not have finalize() method in the implementing class.

JAX-WS Client Model 1-2

- Following are the types of clients in Web Service programming model:

Dynamic proxy client	Dispatch client
• It is also called as static stub client model in JAX-WS. • SEI is used to invoke Web services. Dynamic Proxy Client is used to implement Web services. • In JAX-RPC, tools are used to generate stub client.	• It is also called as dynamic dispatch client model in JAX-WS. • This model is flexible to write XML constructs and is generic.

- Data can be sent by the dispatch client API to construct XML message in any of the following modes:

PAYLOAD mode	MESSAGE mode
• In the javax.xml.ws.Service.Mode.PAYLOAD mode, the dispatch client provides the content of the soap:Body element. • The JAX-WS includes the payload in the soap:Envelope element.	• In the javax.xml.ws.Service.Mode.MESSAGE mode, the dispatch client provides the entire SOAP envelope. • It includes the soap:Envelope, soap:Header, and soap:Body elements.

JAX-WS Client Model 2-2

- Following table lists the methods of javax.xml.ws.Dispatch interface:

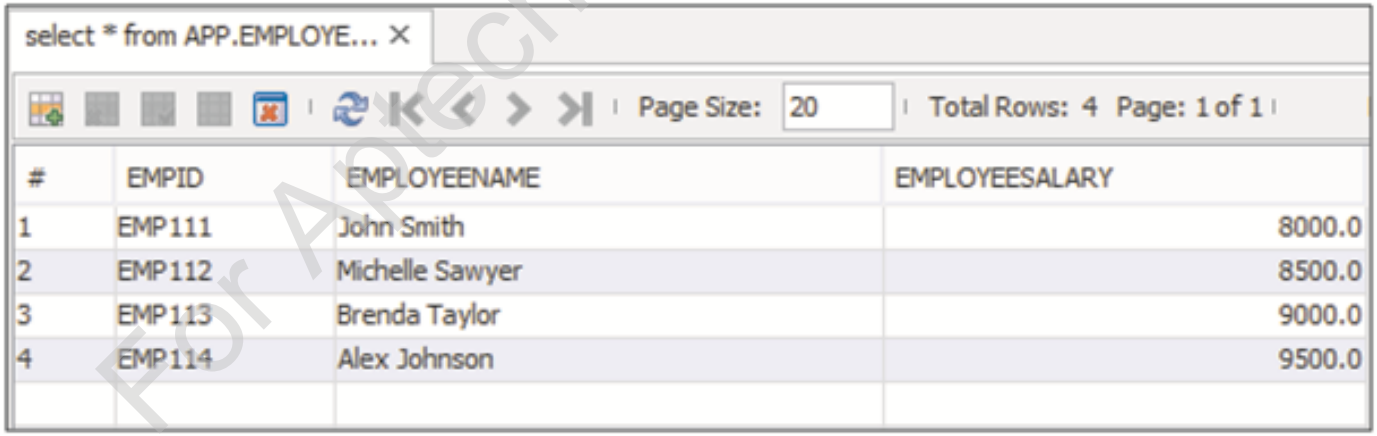
Attribute	Description
T invoke(T msg)	It specifies synchronous invocation of Web service operation. It has a parameter msg. This parameter represents a message object that can have the entire message or payload section of the message. The msg object is marshalled as per the underlying protocol binding.
Response<T> invokeAsync(T msg)	It specifies asynchronous invocation of the Web service operation. It returns to the client without waiting for the response from the Web service method. The response is received by polling.
java.util.concurrent.Future<?> invokeAsync(Tmsg,AsyncHandler<T> handler)	It specifies asynchronous invocation of the Web service operation. It returns to the client without waiting for the response from the Web service method. The response is received by the handler object.

Creating a Simple Web Service Application Using JAX-WS 1-2

- A Web service application can be developed using JAX-WS by annotating the Java class with the @WebService annotation.

- By annotating, the class can be defined as a Web service endpoint. The files required to develop a Web service application using JAX-WS implementation are:
  - Service Interface
  - Service Implementation Class

Consider a scenario where employee details, such as employee ID, employee name, and employee salary are stored in a database table named Employee, as shown in the following figure:



#	EMPID	EMPLOYEENAME	EMPLOYEESALARY
1	EMP111	John Smith	8000.0
2	EMP112	Michelle Sawyer	8500.0
3	EMP113	Brenda Taylor	9000.0
4	EMP114	Alex Johnson	9500.0

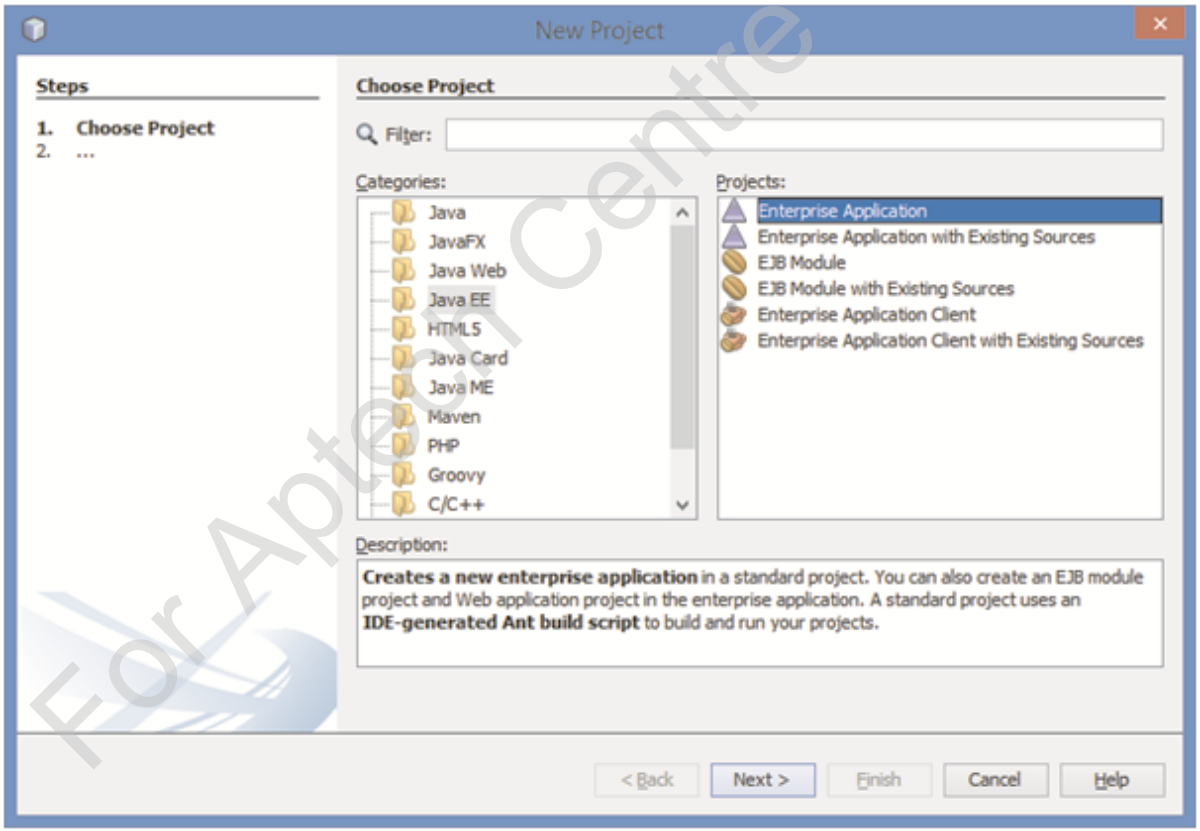
Creating a Simple Web Service Application Using JAX-WS 2-2

- Following are the steps to create a Web service application and client:

1. Create the Web service implementation class.
2. Build, package (war files), and deploy the files. The Glassfish server automatically generates the artifacts required for communicating with the clients.
3. Create the client class.
4. Build and execute the client application.

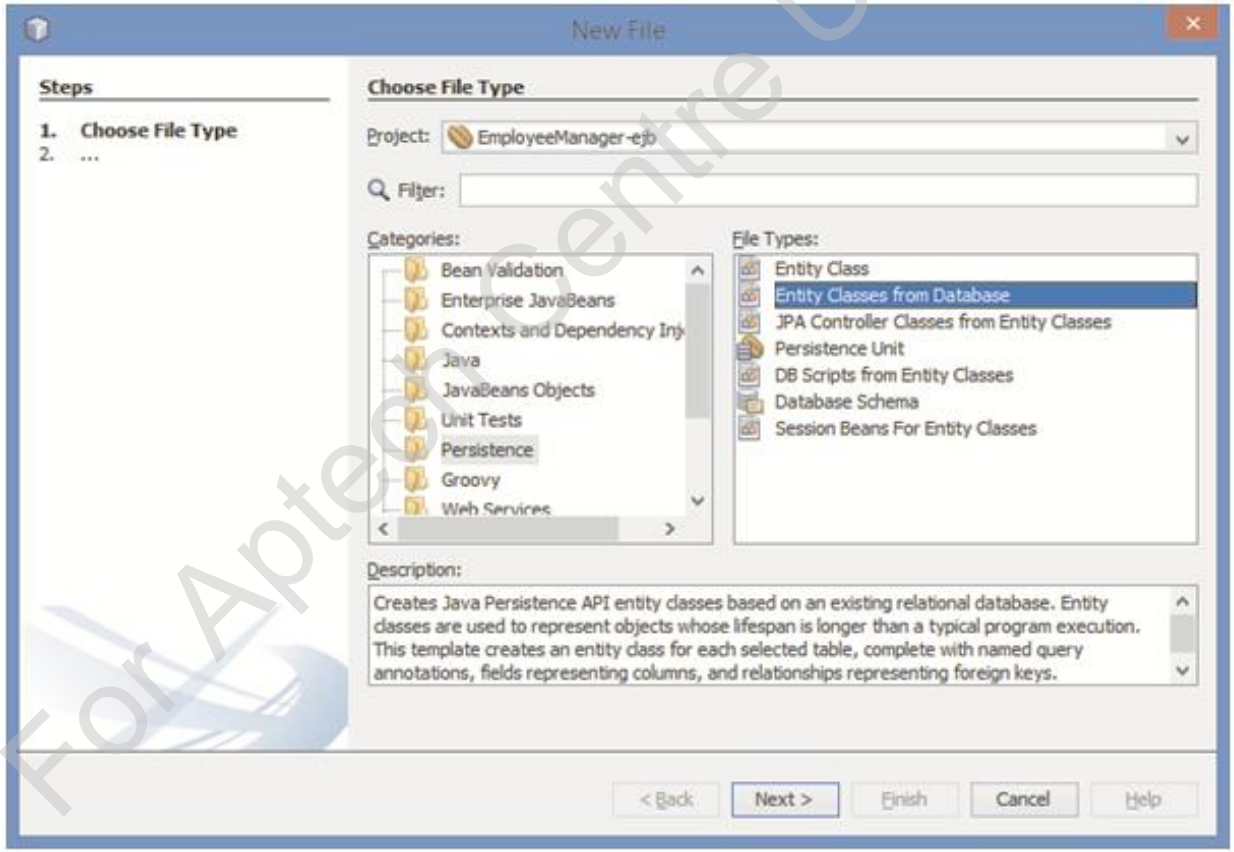
Create a Web Service Implementation Class 1-10

- To create the Web service implementation class, perform the following steps:
- In the NetBeans IDE, click File → New Project → Others and then, in the New Project wizard, select Java EE from the Categories list and Enterprise Application from the Projects list, as shown in the following figure:



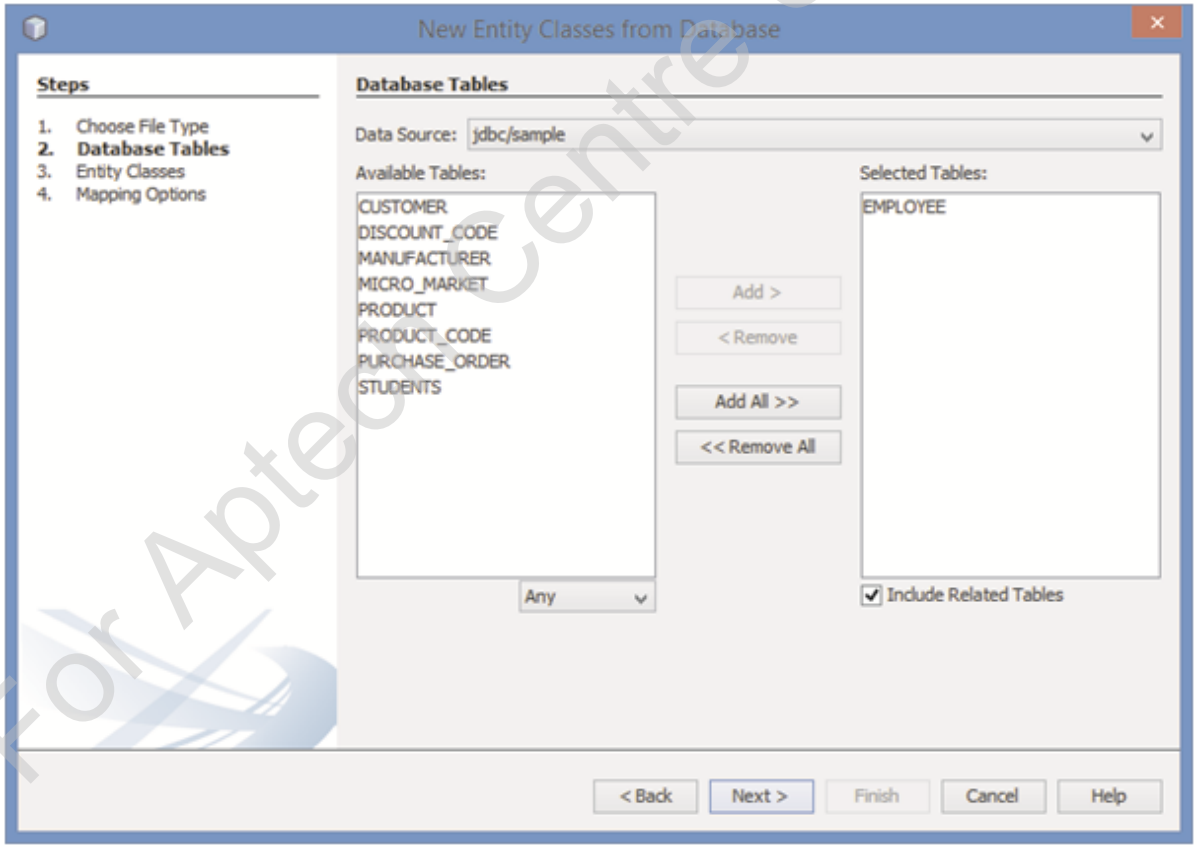
Create a Web Service Implementation Class 2-10

To use entity beans to create the Web service, entity classes need to be created from the Employee database. To do this, right-click EmployeeManager-ejb and then, click New → Other. Then, in the New File wizard, select Persistence in the Categories list and Entity Classes from Database in the File Types list, as shown in the following figure:



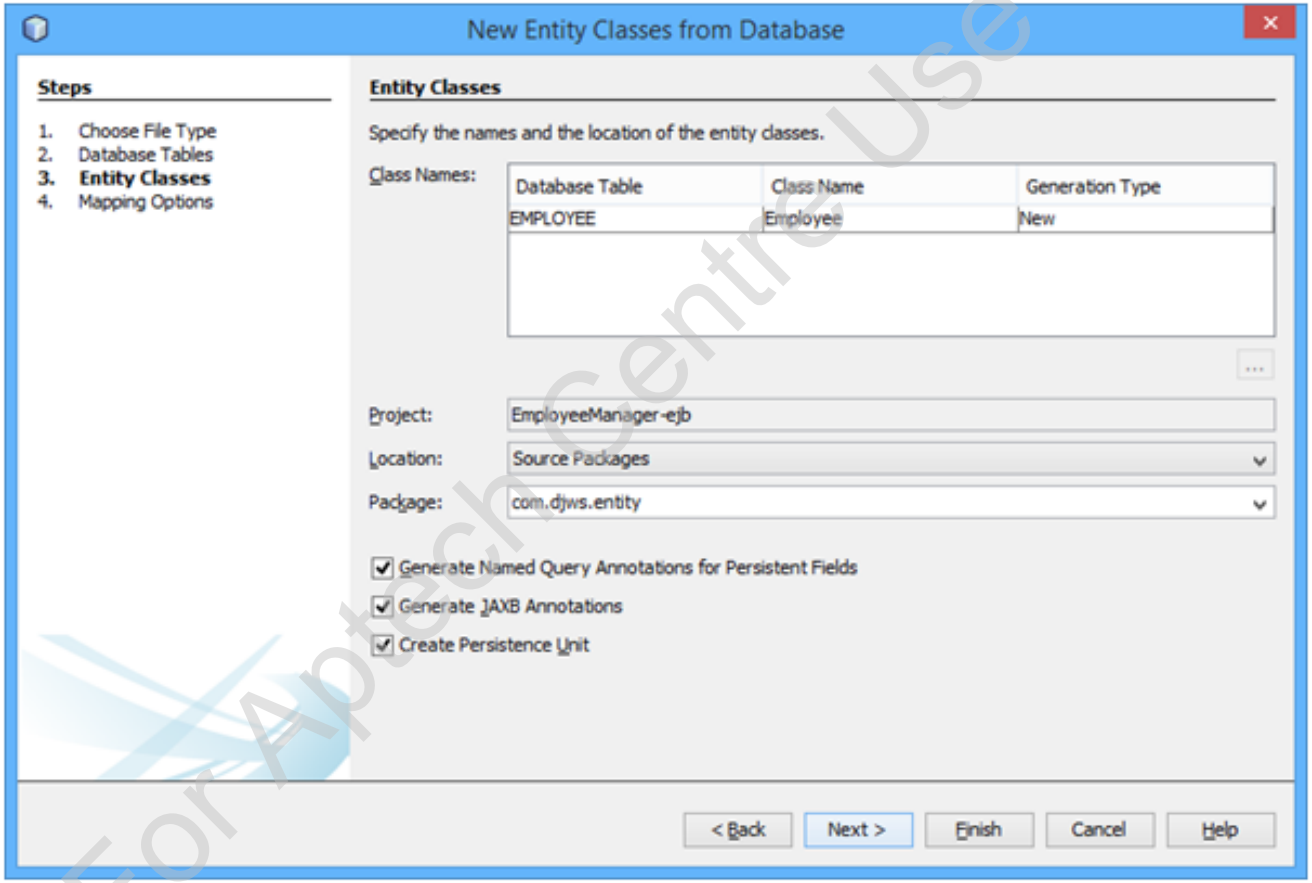
Create a Web Service Implementation Class 3-10

Next, to select the database to use, in the Database Source drop-down list, select jdbc/sample. All the tables associated with the selected source will be displayed in the Available Tables list. Select EMPLOYEE and then, click the Add button to add the table to the Selected Tables list, as shown in the following figure:



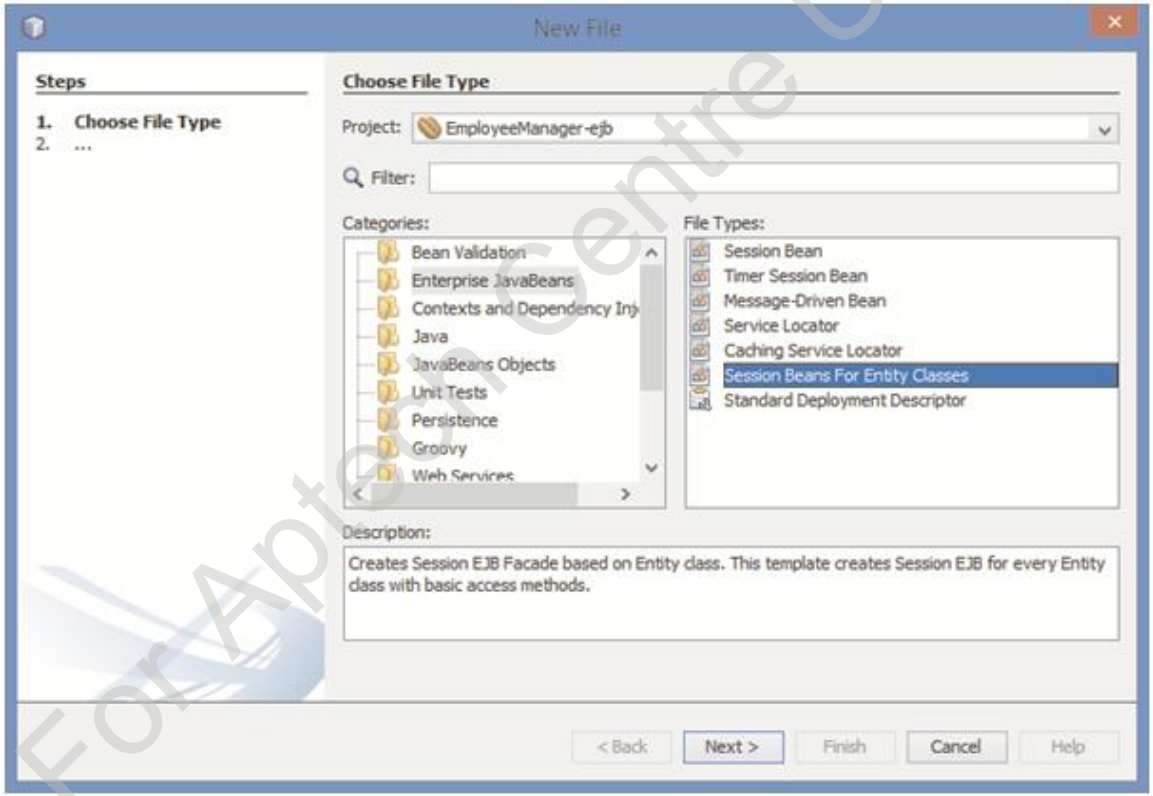
Create a Web Service Implementation Class 4-10

On the next page of the wizard, specify a package for the entity classes and then, click Finish, as shown in the following figure:



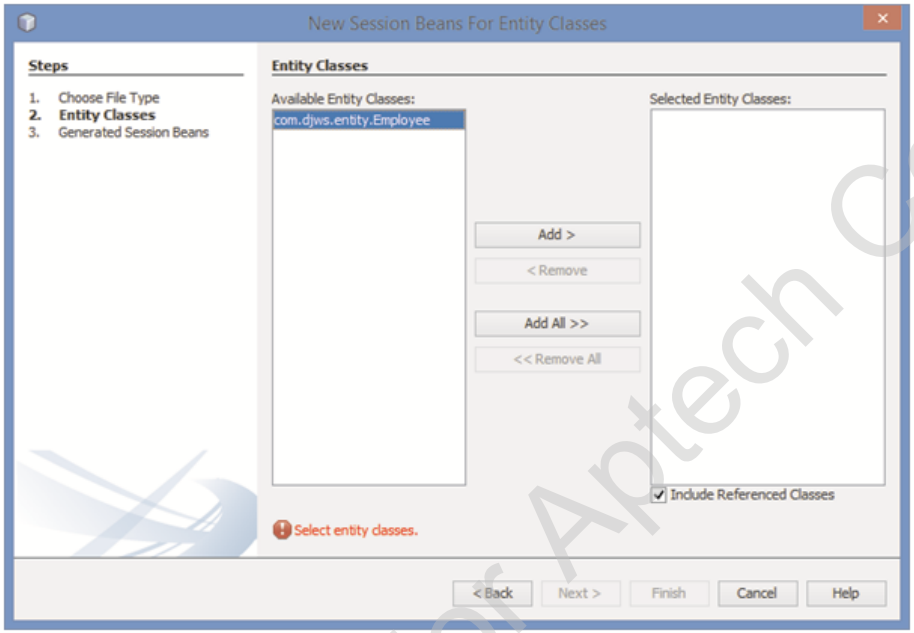
Create a Web Service Implementation Class 5-10

To create session beans from the entity classes, right-click EmployeeManager-ejb and then, click New → Others. Then, in the New File wizard, select Enterprise JavaBeans from the Categories list and Session Beans for Entity Classes from the File Types list, as shown in the following figure:

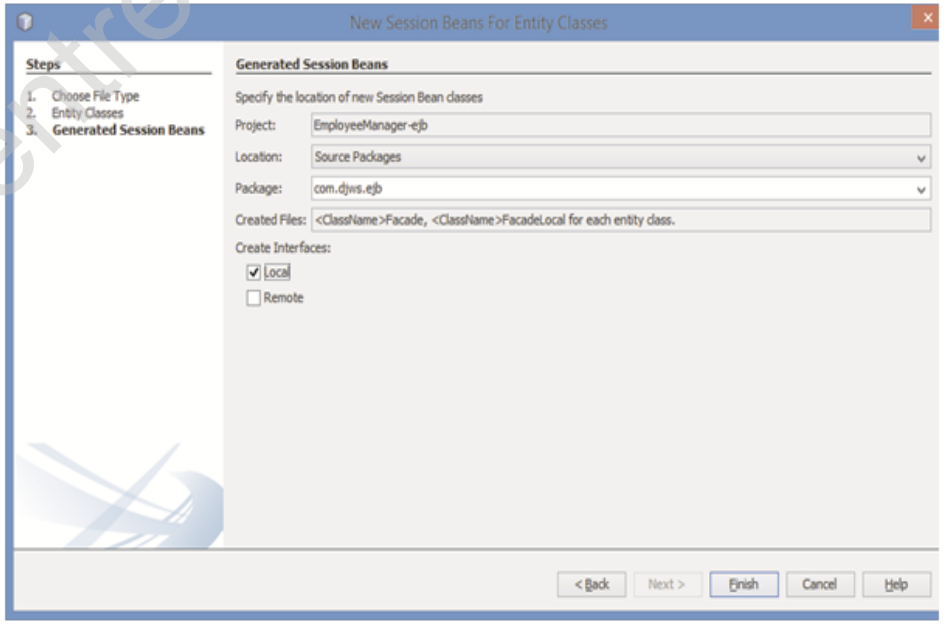


Create a Web Service Implementation Class 6-10

On the next page of the wizard, select the required entity classes from the Available Entity Classes list and then, click the Add button, as shown in the following figure:

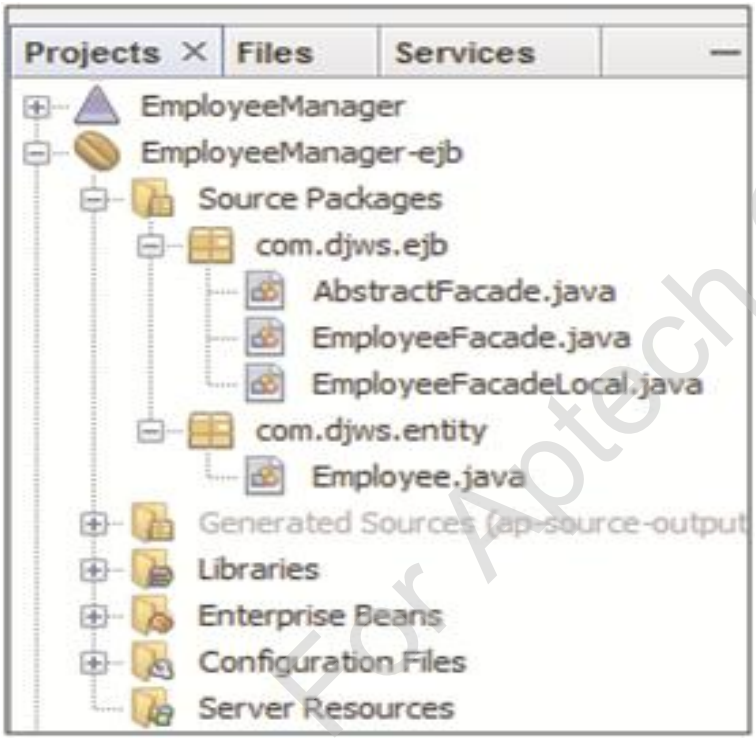


Next, specify a package name for the session beans and select Local to create local interfaces, as shown in the following figure:

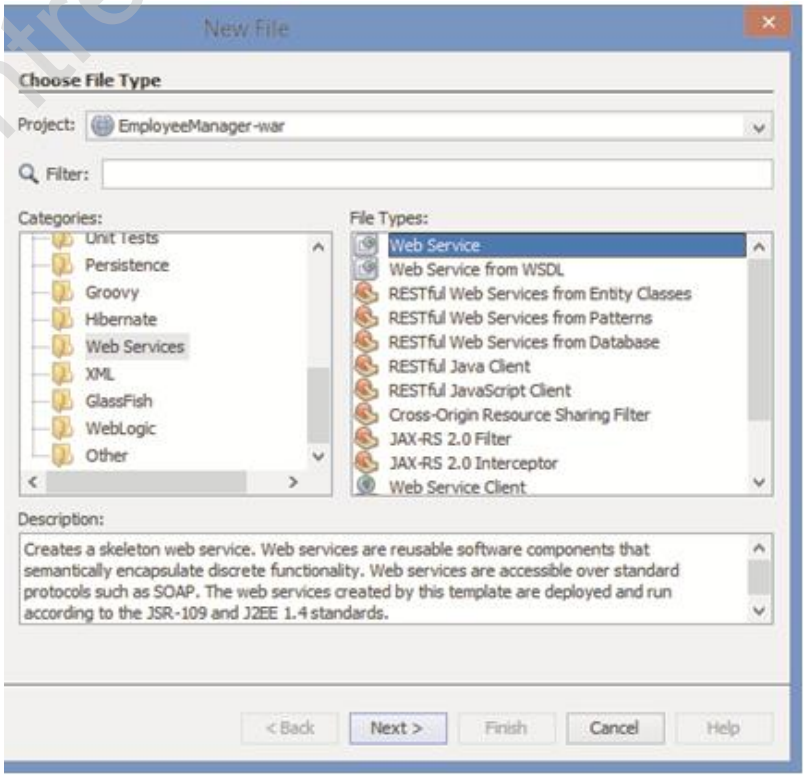


Create a Web Service Implementation Class 7-10

Three Java classes will be created. In this example, the three Java classes are AbstractFacade, EmployeeFacade, and EmployeeFacadeLocal, as shown in the following figure:

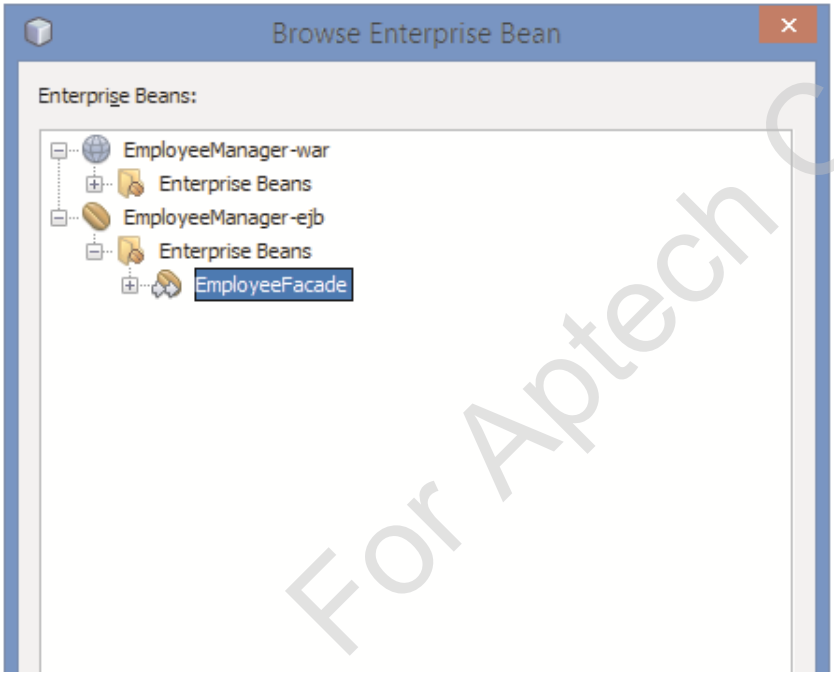


To create a Web Service from the sessions beans, right-click EmployeeManager-war, and then, click New → Other. Then, in the New File wizard, select Web Services in the Categories list and Web Service in the File Types list, as shown in the following figure:



Create a Web Service Implementation Class 8-10

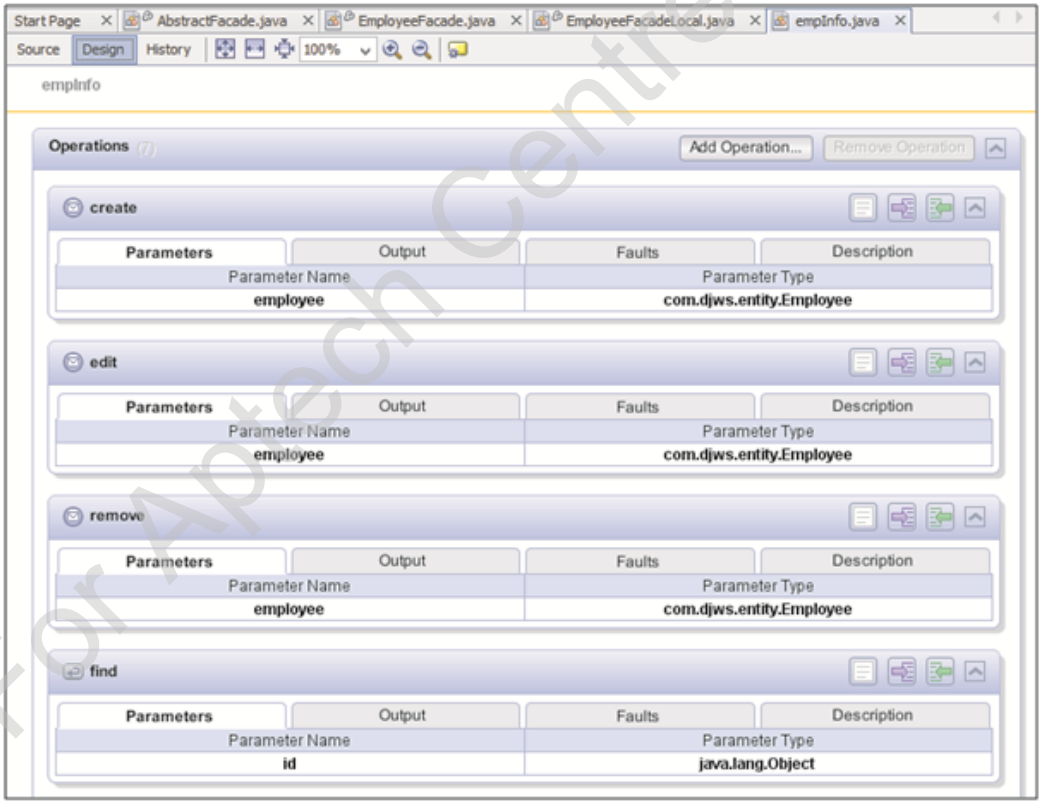
Specify a name for the Web service, a package name for the Web service, select the Create Web Service from Existing Session Bean option, and then, click Browse. In the Browse Enterprise Bean dialog box, select the bean to be used, as shown in the following figure:



Select the Implement Web Service as Stateless Session Bean check box, as shown in the following figure:

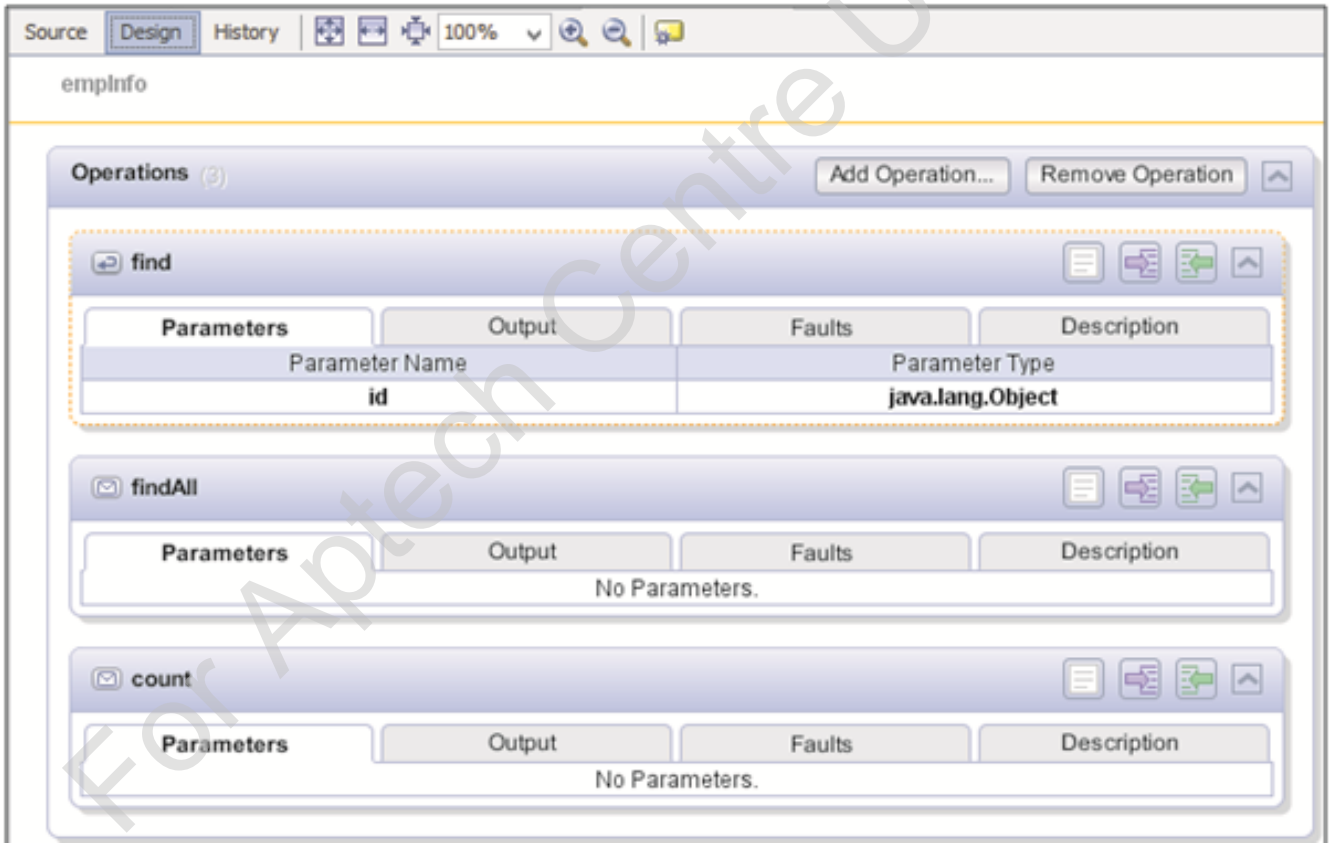
Create a Web Service Implementation Class 9-10

- Alternatively, you can create a session bean from scratch. By default, the Web service is implemented as a stateful session bean. To create a stateless session bean, you need specify explicitly by selecting the check box provided. The .java file for the Web service is created.
- By default, this Web service contains code for several operations, such as create, edit, remove, find, findAll, findRange, and count, which can be viewed by clicking the Design tab as shown in the following figure:



Create a Web Service Implementation Class 10-10

To remove some of these default operations, in the Design tab, click the operation name bar to select the operation and click Remove Operation button. Retain the count, find, and findAll operations, as shown in the following figure:

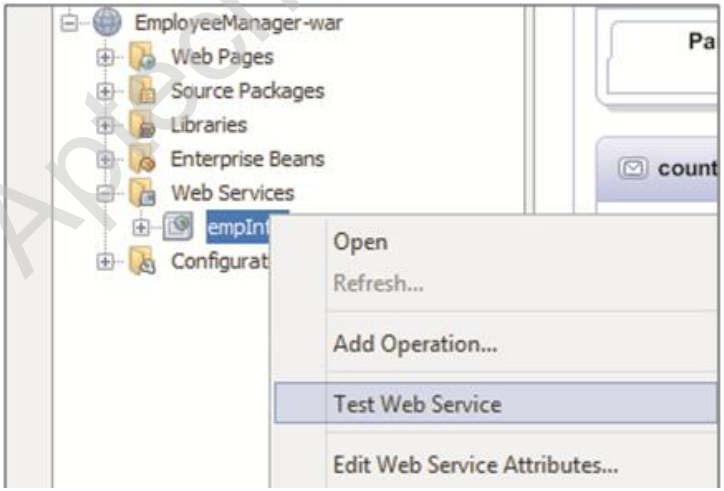


Build, Package, and Deploy the Web Service 1-3

- The generation and deployment of the Web service is done in NetBeans IDE. The steps to do are as follows:

1. Select the project in NetBeans IDE.
2. Right-click to view the context menu.
3. Select Deploy from the context menu.

The command Deploy builds and packages the Web application into the WAR file. Then, it deploys the file on the application server, that is, GlassFish Server. To test the Web Service, right-click the Web Service and then, in the context menu, click Test Web Service, as shown in the following figure:

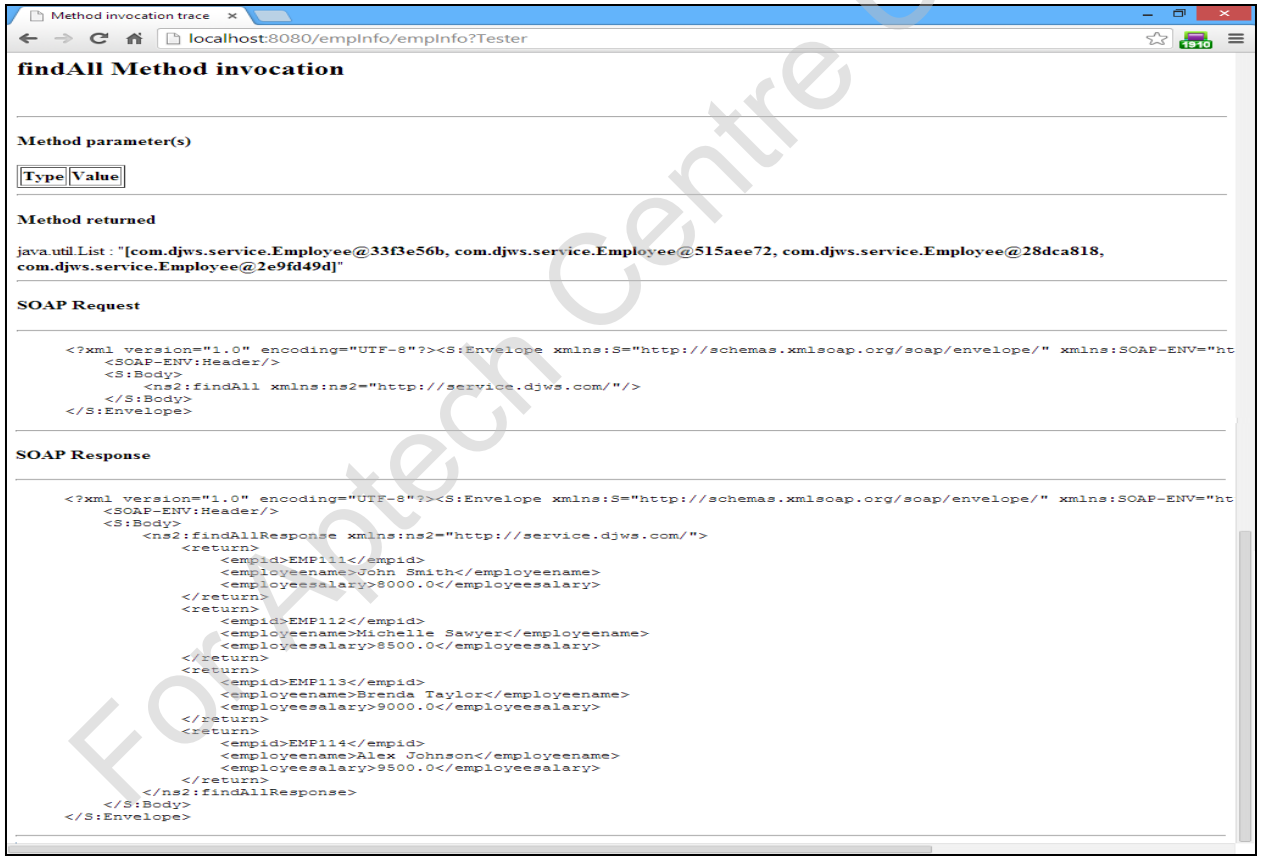


Build, Package, and Deploy the Web Service 2-3

The tester Web page appears as shown in the following figure:

Build, Package, and Deploy the Web Service 3-3

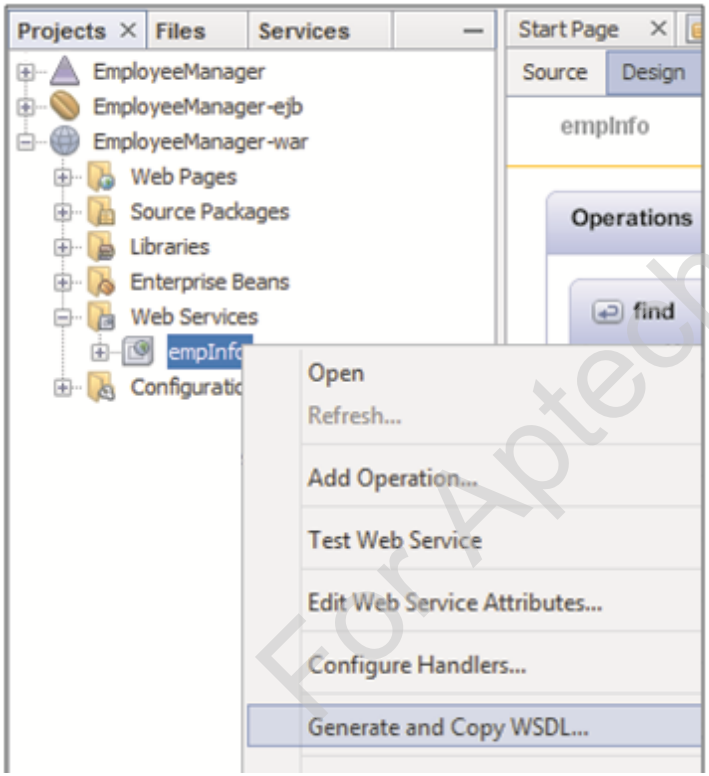
- The Web Service Tester displays the operations that were retained—count, find, and findAll.
- The count operation will display the number of records in the Employee table. The find operation will display the details of the employee whose employee ID has been specified in the box.
- The findAll operation will display the details of all employees stored in the Employee table. Following figure shows the output of the findAll operation:



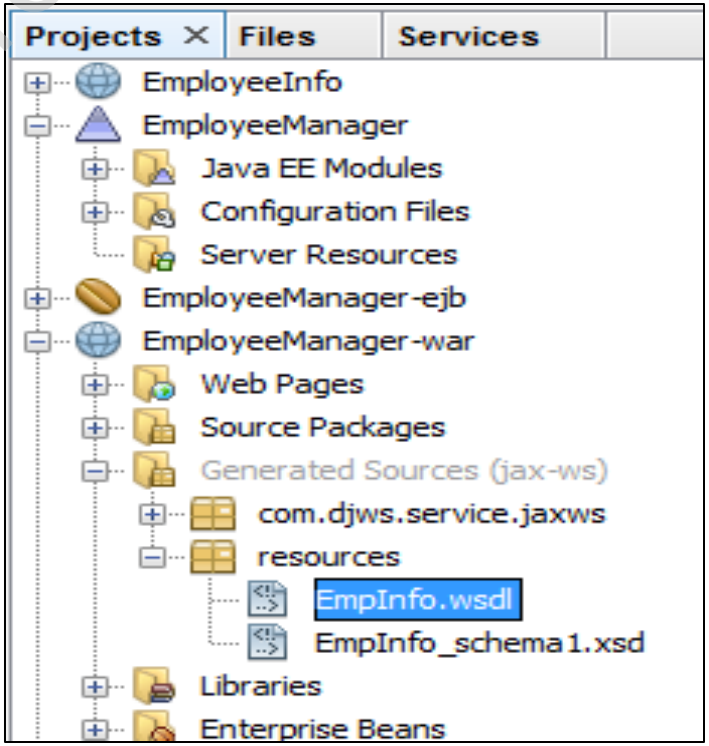
Create Web Client 1-6

- A Web client can be created using JSP, Servlet, and so on. The WSDL file is generated based on the Web service.

The WSDL can also be copied to the project. To do this, right-click the Web service and then, click Generate and Copy WSDL as shown in the following figure:



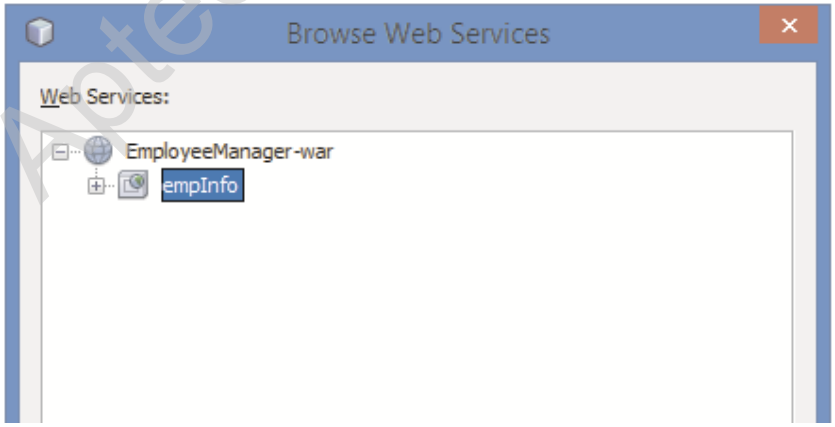
A new folder is created in the project named Generated Sources (jax-ws). To view the WSDL file, expand Generated Sources (jax-ws) and then expand resources, as shown in the following figure:



Create Web Client 2-6

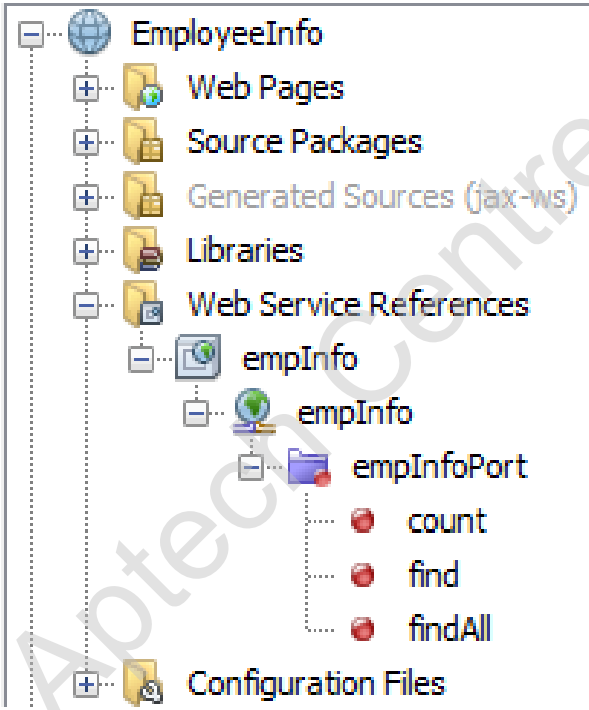
The WSDL file is used to generate the Web service client.
Perform the following steps to generate the Web service client:

1. • Create a new Web Application project in NetBeans IDE.
2. • Ensure that the Glassfish Server is selected and the Context Path is set.
3. • Right-click the project and then click New → Others and then in the New File Wizard, select Web Services in the Categories list and Web Service Client in the File Types list.
4. • To specify the project where the WSDL file is located, click Browse next to the Project box.
5. • Browse to the Web service for which the Web service client is being created, as shown in the following figure:



Create Web Client 3-6

A Web Service References folder is created in the project. This folder contains the method exposed by the Web service, as shown in the following figure:



Create Web Client 4-6

6. • Add a new JSP file by right-clicking the project and then clicking New → JSP.

7. • Name the JSP file as index.

Drag the find() method from the Web Service Reference folder to the index.jsp file and place it after the h1 heading. The updated index.jsp file appears, as shown in the following figure:

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title>JSP Page</title>

    </head>
    <body>
        <h1>Hello World</h1>    <%-- start web service invocation --%><hr/>
    <%
    try {
        com.djws.service.EmpInfo_Service service = new com.djws.service.EmpInfo_Service();
        com.djws.service.EmpInfo port = service.getEmpInfoPort();
         // TODO initialize WS operation arguments here
                java.lang.Object id = null;
        // TODO process result here
        com.djws.service.Employee result = port.find(id);
    } catch (Exception ex) {
         // TODO handle custom exceptions here
    }
    %>
    <%-- end web service invocation --%><hr/>

    </body>
</html>
```

Create Web Client 5-6

Update the title of the JSP page and h1 heading, add an input text box for entering the employee ID and a button, add the code to use the Web service to retrieve and display the relevant employee details and add the exception handling code in the catch block as shown in the following code snippet:

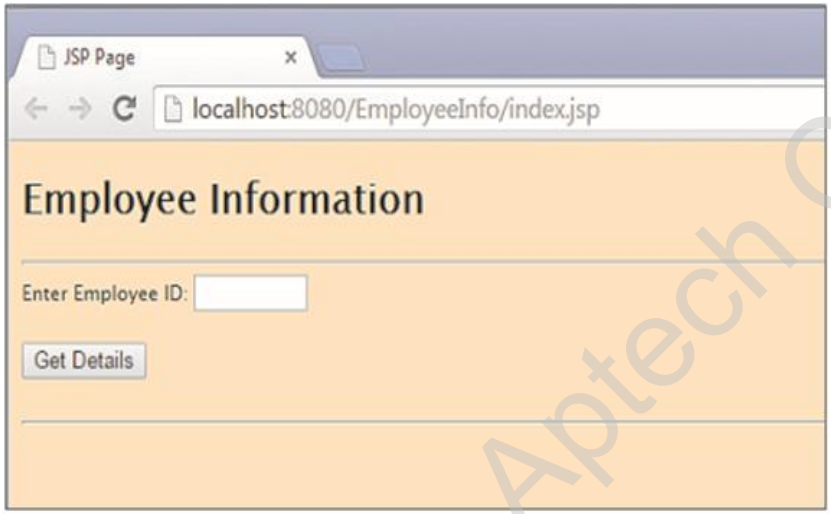
```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title>JSP Page</title>
     <link href="style.css" rel="stylesheet" type="text/css"/>
    </head>
    <body>
        <h1>Employee Information</h1>
<%-- start Web service invocation --%><hr/>
        <form action="index.jsp">
        Enter Employee ID:
        <input type="text" id="txtbox" name="empid" size="6"
maxlength="6">
        <br></br>
        <input type="submit" value="Get Details">
        <br></br>
<%
```

Create Web Client 6-6

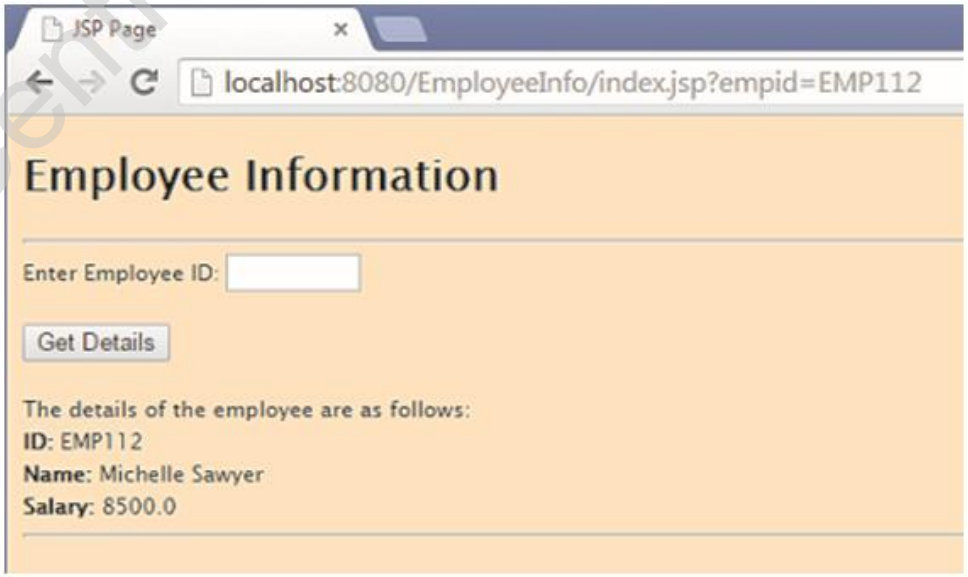
```
try {
com.djws.service.EmpInfo_Service service = new
com.djws.service.EmpInfo_Service();
          com.djws.service.EmpInfo port = service.getEmpInfoPort();

// TODO initialize WS operation arguments here
          if(request.getParameter("empid")!=null){
java.lang.String id =request.getParameter("empid");

          // TODO process result here
          com.djws.service.Employee emp = port.find(id);
          out.println("The details of the employee are as follows:");
      out.println("<br>");
      out.println("<b>ID: </b>" + emp.getEmpid());
      out.println("<br>");
      out.println("<b>Name: </b>" + emp.getEmployeename());
      out.println("<br>");
      out.println("<b>Salary: </b>" + emp.getEmployeesalary());
            }
       }
catch (Exception ex) {
          out.println("Exception :" + ex);
       }
    %>
    <%-- end Web service invocation --%><hr/>
 </form>     </body>   </html>
```

Build Package Deploy the JSP Client

To build, package, and deploy the client application, right-click the project name and select Run from the context menu. The output of the client application is as shown in the following figure:



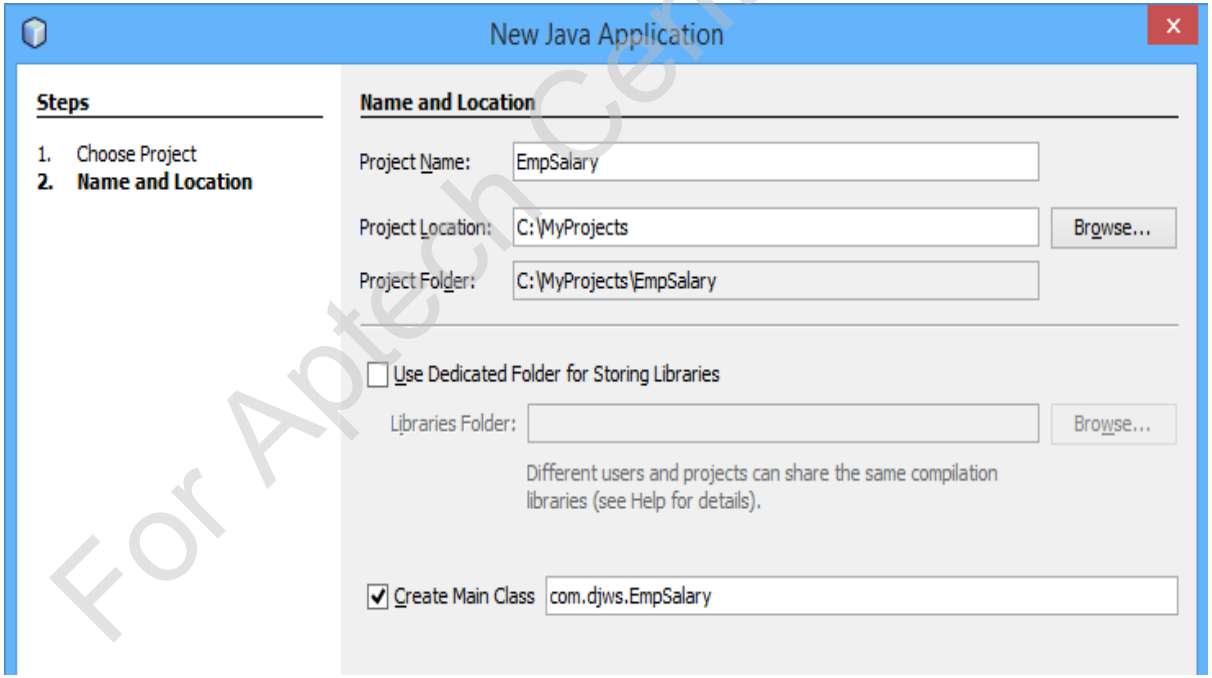
When an employee ID is specified and the Get Details button is clicked, the name and salary of the employee whose ID has been specified is displayed on the Web page as shown in the following figure:



Create an Application Client 1-8

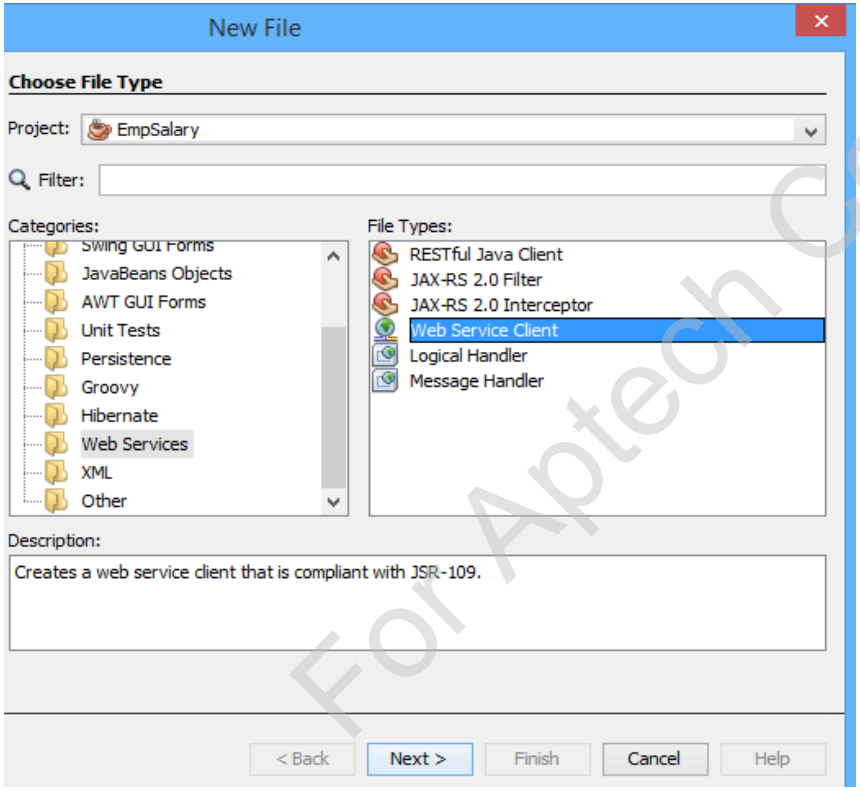
You can also create a Java application client for the Web service. To do this, perform the following steps:

1. Create a new Java Application project in NetBeans IDE by clicking File → New Project and then, in the New Project wizard, select Java in the Categories list and Java Application in the Projects list.
2. Specify a name and location for the project and ensure that the Create Main Class check box is selected as shown in the following figure:

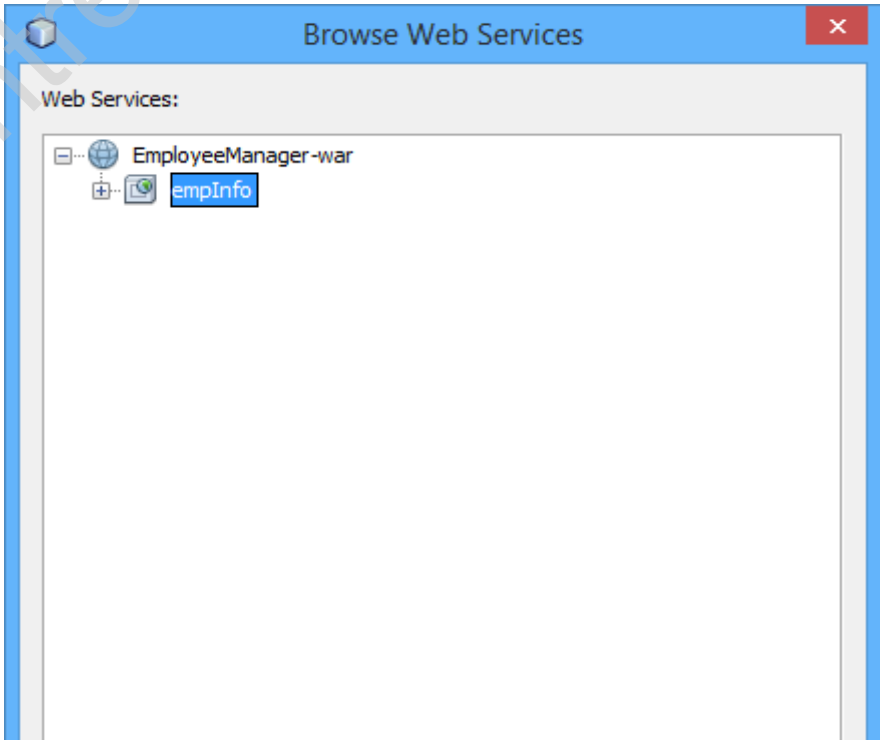


Create an Application Client 2-8

3. To create a Web service client, right-click the Java Application project and then click New → Others. Then, in the New File wizard, select Web Services in the Categories list and Web Service Client in the File Types list, as shown in the following figure:

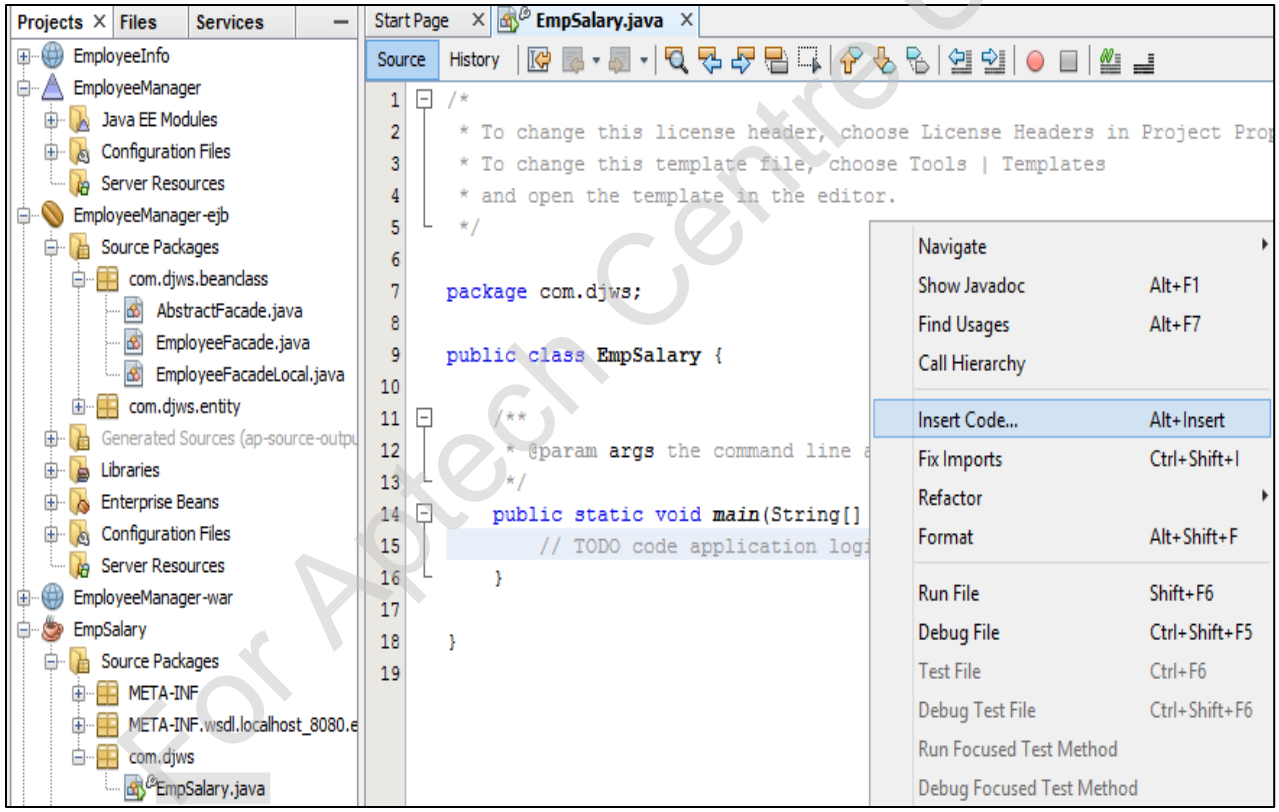


4. To specify the project that contains the Web service, in the New Web Service Client wizard, click Browse, next to the Project box, and then in the Browse Web Service dialog box, select the Web service for which the client is being created, as shown in the following figure:



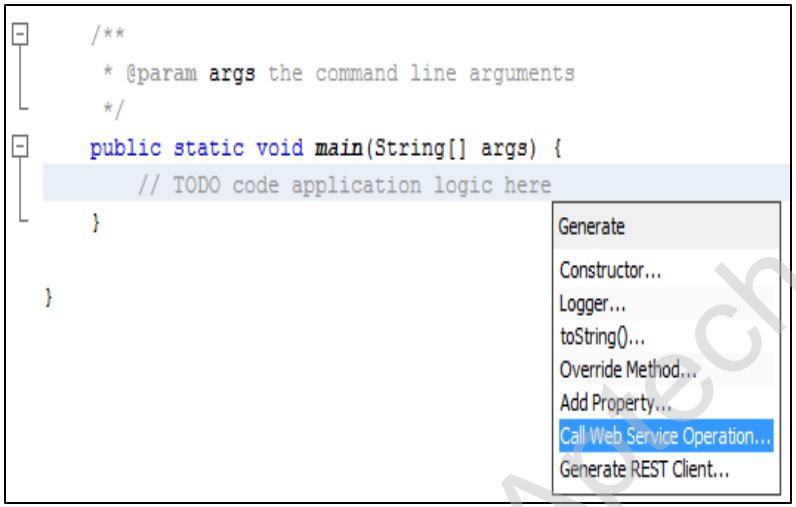
Create an Application Client 3-8

5. The Web Service code will be generated. Build the Web service client project to compile all the generated resources.
6. Open the EmpSalary.java file, right-click inside the main() method, and then click Insert Code as shown in the following figure:

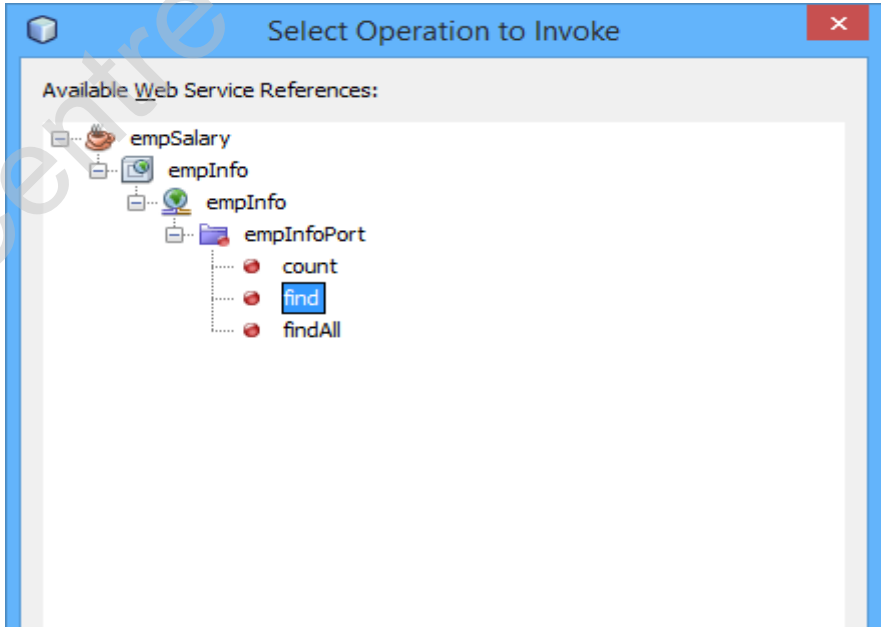


Create an Application Client 4-8

7. In the context menu that appears, click Call Web Service Operation, as shown in the following figure:



8. In the Select Operation to Invoke dialog box, select the find() method, as shown in the following figure:



Create an Application Client 5-8

The IDE adds the code required to invoke the find() method, as shown in the following figure:

```
11
12  public class EmpSalary {
13
14      /**
15       * @param args the command line arguments
16       */
17      public static void main(String[] args) {
18          // TODO code application logic here
19      }
20
21      private static Employee find(java.lang.Object id) {
22          com.djws.service.EmpInfo_Service service = new com.djws.service.EmpInfo_Ser
23          com.djws.service.EmpInfo port = service.getEmpInfoPort();
24          return port.find(id);
25      }
26
27  }
```

Create an Application Client 6-8

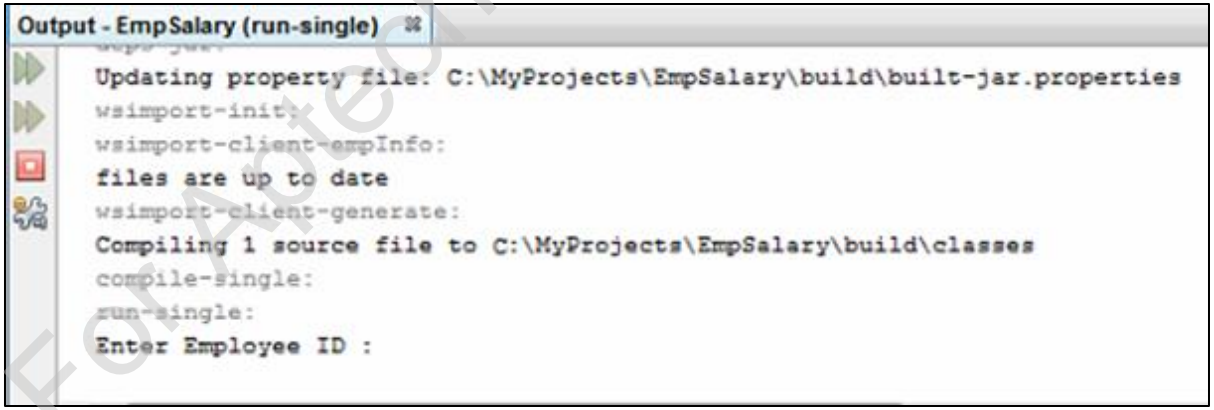
9. To accept user input and display the output, add the code given as shown in code snippet, in the main() method of the EmpSalary class.

```
...
import java.util.scanner;
...
{
public static void main(String[] args)
{
    String empId = null;
    Scanner in = new Scanner(System.in);
    System.out.println("Enter Employee Id :");
    empId = in.nextLine();
    Employee emp = new Employee();
    emp = find (empId);
    System.out.println("Employee ID: " + emp.getEmpid());
    System.out.println("Employee Name: " +
emp.getEmployeename());
    System.out.println("Employee Salary: " +
                        emp.getEmployeesalary());
}
```

Create an Application Client 7-8

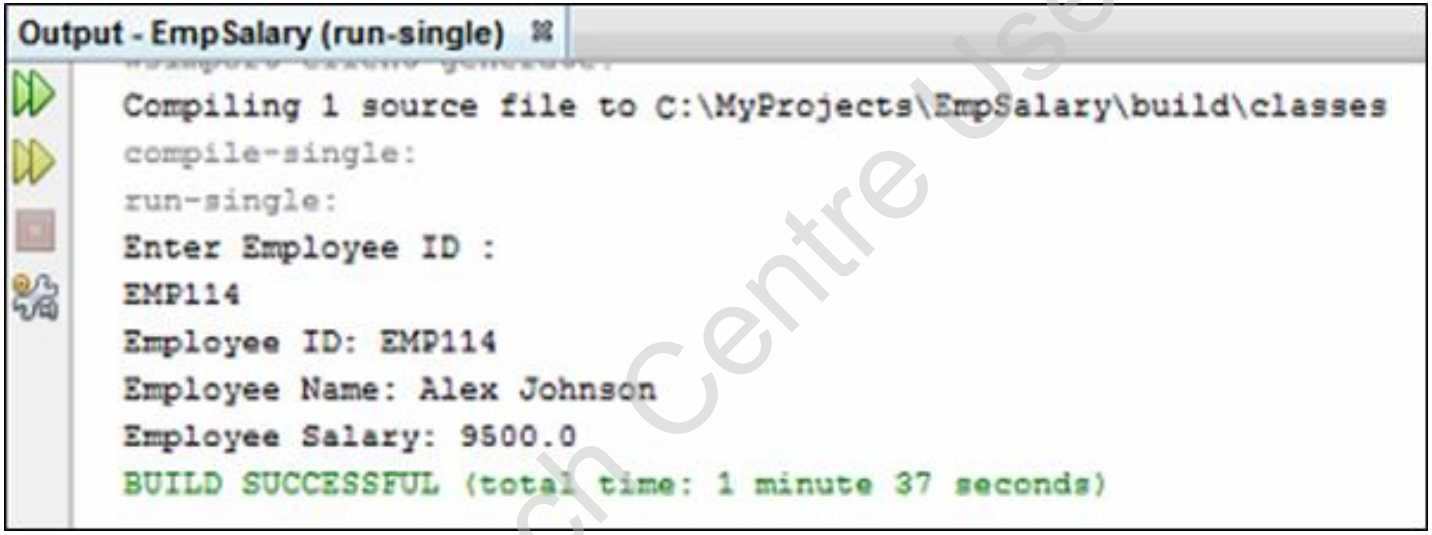
```
    private static Employee find(java.lang.String id)
    {
        com.djws.EmpInfo_Service service = new
             com.djws.EmpInfo_Service();
        com.djws.EmpInfo port = service.getEmpInfoPort();
        return port.find(id);
    }
    }
```

10. Build and run the EmpSalary project. The user will be prompted to enter the employee ID as shown in the following figure:



Create an Application Client 8-8

11. Specify employee ID EMP114 and then, press ENTER. The employee name and employee salary is displayed as shown in the following figure:



Summary

- Web services can be developed using annotations and Plain Old Java Objects (POJOs). They are available on JEE platform, which provides the right environment for easy development and deployment of Web services.
- There are two types of Web services. They are 'Big' Web services and RESTful Web services.
- JAX-WS supports annotations that make development of Web service applications simple and easy.
- JAX-WS data binding and parsing are based on the JAXB and StAX.
- JAX-WS architecture is based on the generation of dynamic proxy class instance, which is responsible to send/receive SOAP request/response on the client/server.
- The two types of service endpoint implementations supported by JAX-WS are the standard JavaBeans service endpoint interface and a new Provider interface.
- The different types of clients in Web service client programming model supported by JAX-WS are Dynamic proxy client and Dispatch client. They support synchronous and asynchronous invocations on the client-side.