

Practical No: 05

Aim: Show the implementation of web services.

What Are Web Services?

Web services are client and server applications that communicate over the World Wide Web's (WWW) HyperText Transfer Protocol (HTTP). As described by the World Wide Web Consortium (W3C), web services provide a standard means of interoperating between software applications running on a variety of platforms and frameworks. Web services are characterized by their great interoperability and extensibility, as well as their machine-processable descriptions, thanks to the use of XML. Web services can be combined in a loosely coupled way to achieve complex operations. Programs providing simple services can interact with each other to deliver sophisticated added-value services.

Types of Web Services:

On the conceptual level, a service is a software component provided through a network-accessible endpoint. The service consumer and provider use messages to exchange invocation request and response information in the form of self-containing documents that make very few assumptions about the technological capabilities of the receiver.

On a technical level, web services can be implemented in various ways. The two types of web services can be distinguished as “big” web services and “RESTful” web services.

1) “Big” Web Services:

In Java EE 6, JAX-WS provides the functionality for “big” web services. Big web services use XML messages that follow the Simple Object Access Protocol (SOAP) standard, an XML language defining a message architecture and message formats. Such systems often contain a machine-readable description of the operations offered by the service, written in the Web Services Description Language (WSDL), an XML language for defining interfaces syntactically.

The SOAP message format and the WSDL interface definition language have gained widespread adoption. Many development tools, such as NetBeans IDE, can reduce the complexity of developing web service applications.

A SOAP-based design must include the following elements.

- ❑ A formal contract must be established to describe the interface that the web service offers. WSDL can be used to describe the details of the contract, which may include messages, operations, bindings, and the location of the web service. You may also process SOAP messages in a JAX-WS service without publishing a WSDL.
- ❑ The architecture must address complex nonfunctional requirements. Many web service specifications address such requirements and establish a common vocabulary for them. Examples include transactions, security, addressing, trust, coordination, and so on.

- ❑ The architecture needs to handle asynchronous processing and invocation. In such cases, the infrastructure provided by standards, such as Web Services Reliable Messaging (WSRM), and APIs, such as JAX-WS, with their client-side asynchronous invocation support, can be leveraged out of the box.

2) **RESTful Web Services:**

In Java EE 6, JAX-RS provides the functionality for Representational State Transfer (RESTful) web services. REST is well suited for basic, ad hoc integration scenarios. RESTful web services, often better integrated with HTTP than SOAP-based services are, do not require XML messages or WSDL service-API definitions.

Project Jersey is the production-ready reference implementation for the JAX-RS specification. Jersey implements support for the annotations defined in the JAX-RS specification, making it easy for developers to build RESTful web services with Java and the Java Virtual Machine (JVM).

Because RESTful web services use existing well-known W3C and Internet Engineering Task Force (IETF) standards (HTTP, XML, URI, MIME) and have a lightweight infrastructure that allows services to be built with minimal tooling, developing RESTful web services is inexpensive and thus has a very low barrier for adoption. You can use a development tool such as NetBeans IDE to further reduce the complexity of developing RESTful web services.

A RESTful design may be appropriate when the following conditions are met.

- ❑ The web services are completely stateless. A good test is to consider whether the interaction can survive a restart of the server.
- ❑ A caching infrastructure can be leveraged for performance. If the data that the web service returns is not dynamically generated and can be cached, the caching infrastructure that web servers and other intermediaries inherently provide can be leveraged to improve performance. However, the developer must take care because such caches are limited to the HTTP GET method for most servers.
- ❑ The service producer and service consumer have a mutual understanding of the context and content being passed along. Because there is no formal way to describe the web services interface, both parties must agree out of band on the schemas that describe the data being exchanged and on ways to process it meaningfully. In the real world, most commercial applications that expose services as RESTful implementations also distribute so-called value-added toolkits that describe the interfaces to developers in popular programming languages.
- ❑ Bandwidth is particularly important and needs to be limited. REST is particularly useful for limited-profile devices, such as PDAs and mobile phones, for which the overhead of headers and additional layers of SOAP elements on the XML payload must be restricted.
- ❑ Web service delivery or aggregation into existing web sites can be enabled easily with a RESTful style. Developers can use such technologies as JAX-RS and Asynchronous JavaScript with XML (AJAX) and such toolkits as Direct Web Remoting (DWR) to consume the services in their web applications. Rather than starting from scratch, services can be exposed with XML and consumed by HTML pages without significantly

refactoring the existing web site architecture. Existing developers will be more productive because they are adding to something they are already familiar with rather than having to start from scratch with new technology.

Deciding Which Type of Web Service to Use:

Basically, you would want to use RESTful web services for integration over the web and use big web services in enterprise application integration scenarios that have advanced quality of service (QoS) requirements.

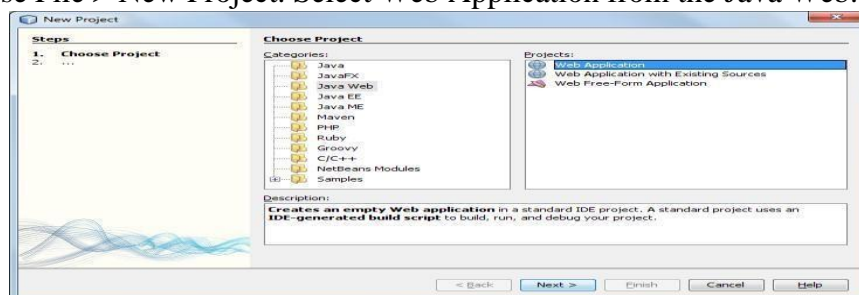
- ❑ **JAX-WS:** addresses advanced QoS requirements commonly occurring in enterprise computing. When compared to JAX-RS, JAX-WS makes it easier to support the WS-* set of protocols, which provide standards for security and reliability, among other things, and interoperate with other WS-* conforming clients and servers.
- ❑ **JAX-RS:** makes it easier to write web applications that apply some or all of the constraints of the REST style to induce desirable properties in the application, such as loose coupling (evolving the server is easier without breaking existing clients), scalability (start small and grow), and architectural simplicity (use off-the-shelf components, such as proxies or HTTP routers). You would choose to use JAX-RS for your web application because it is easier for many types of clients to consume RESTful web services while enabling the server side to evolve and scale. Clients can choose to consume some or all aspects of the service and mash it up with other web-based services.

Practical 5A: Implementing “Big” Web Service.

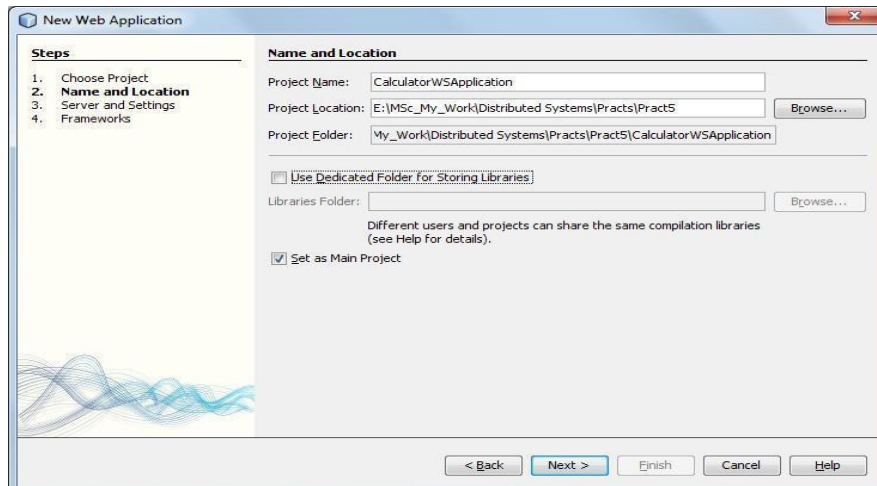
1) Creating a Web Service

A. Choosing a Container:

1. Choose File > New Project. Select Web Application from the Java Web.



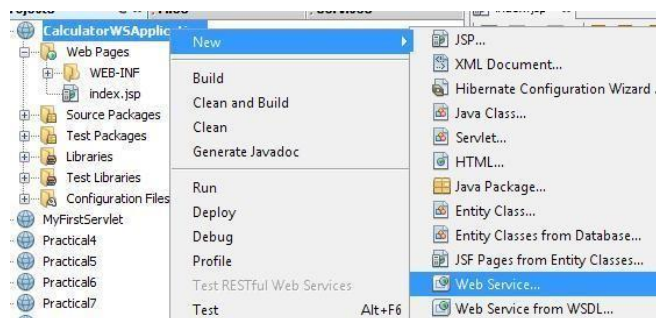
2. Name the project CalculatorWSApplication. Select a location for the project. Click Next.



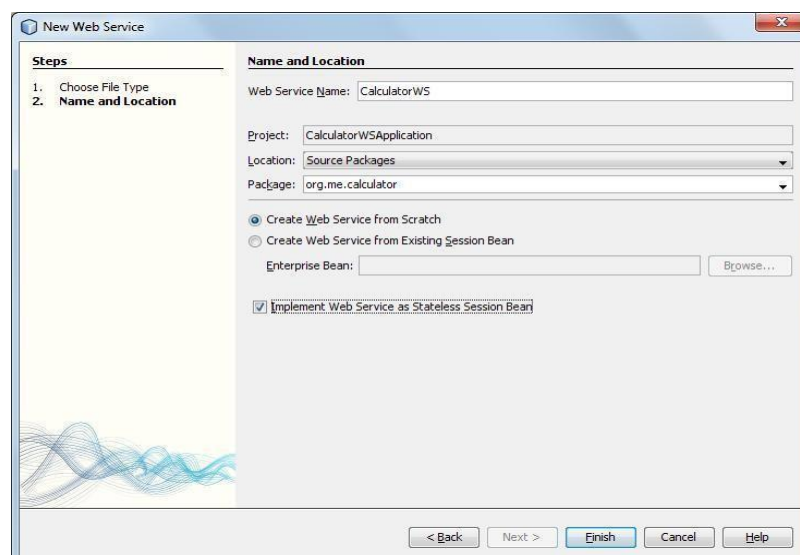
3. Select your server and Java EE version and click Finish.

B. Creating a Web Service from a Java Class

1. Right-click the CalculatorWSApplication node and choose New > Web Service.



2. Name the web service CalculatorWS and type org.me.calculator in Package. Leave Create Web Service from Scratch selected. If you are creating a Java EE 6 project on GlassFish or WebLogic, select Implement Web Service as a Stateless Session Bean.



- Click Finish. The Projects window displays the structure of the new web service and the source code is shown in the editor area.

2) **Adding an Operation to the Web Service**

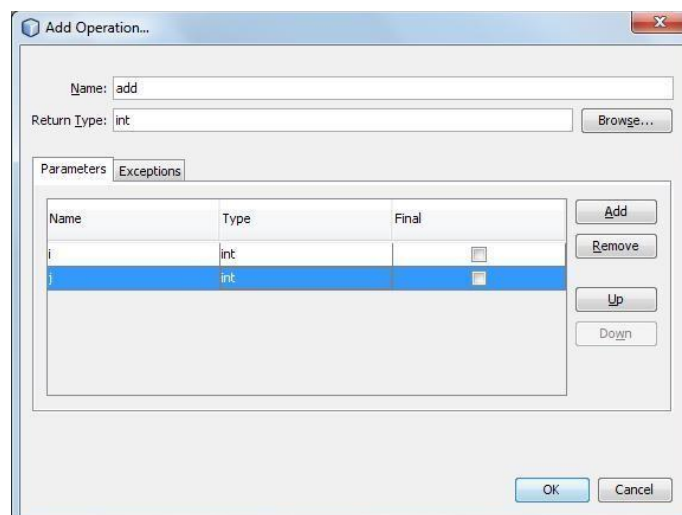
The goal of this exercise is to add to the web service an operation that adds two numbers received from a client. The NetBeans IDE provides a dialog for adding an operation to a web service. You can open this dialog either in the web service visual designer or in the web service context menu.

A. **To add an operation to the web service:**

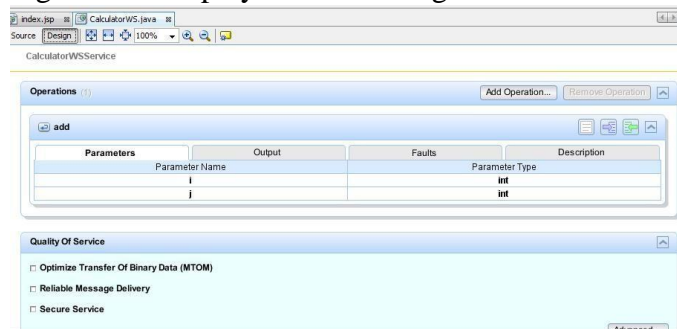
- Change to the Design view in the editor.



- Click Add Operation in either the visual designer or the context menu. The Add Operation dialog opens.
- In the upper part of the Add Operation dialog box, type add in Name and type int in the Return Type drop-down list.
- In the lower part of the Add Operation dialog box, click Add and create a parameter of type int named i.
- Click Add again and create a parameter of type int called j. You now see the following:



6. Click OK at the bottom of the Add Operation dialog box. You return to the editor.
7. The visual designer now displays the following:



8. Click Source. And code the following.

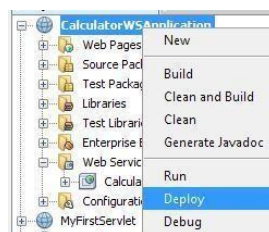
```
@WebMethod(operationName = "add")
public int add(@WebParam(name = "i") int i, @WebParam(name = "j") int j)
{
    int k = i + j;
    return k;
}
```

3) Deploying and Testing the Web Service

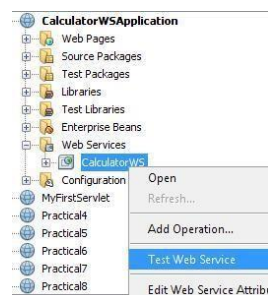
After you deploy a web service to a server, you can use the IDE to open the server's test client, if the server has a test client. The GlassFish and WebLogic servers provide test clients.

A. To test successful deployment to a GlassFish or WebLogic server:

1. Right-click the project and choose Deploy. The IDE starts the application server, builds the application, and deploys the application to the server



2. In the IDE's Projects tab, expand the Web Services node of the CalculatorWSApplication project. Right-click the CalculatorWS node, and choose Test Web Service.



- The IDE opens the tester page in your browser, if you deployed a web application to the GlassFish server.
- If you deployed to the GlassFish server, type two numbers in the tester page, as shown below:

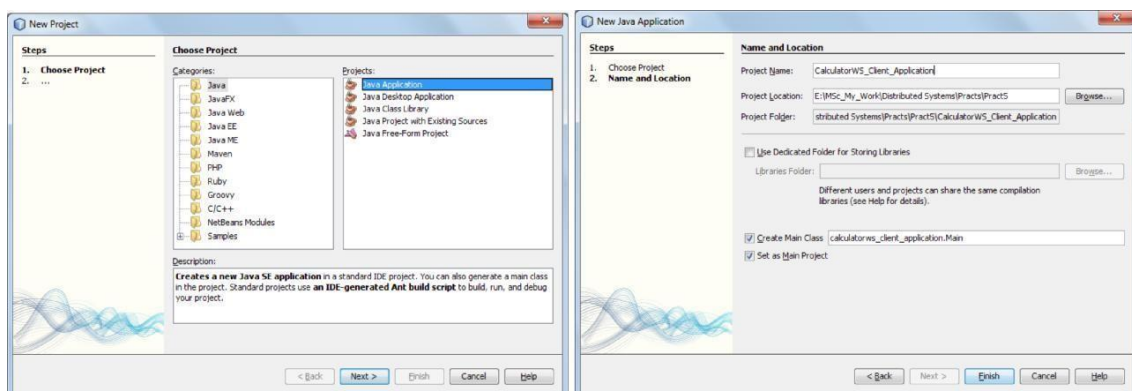
- The sum of the two numbers is displayed:

4) Consuming the Web Service

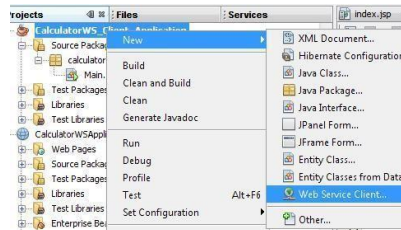
Now that you have deployed the web service, you need to create a client to make use of the web service's add method.

1. Client: Java Class in Java SE Application

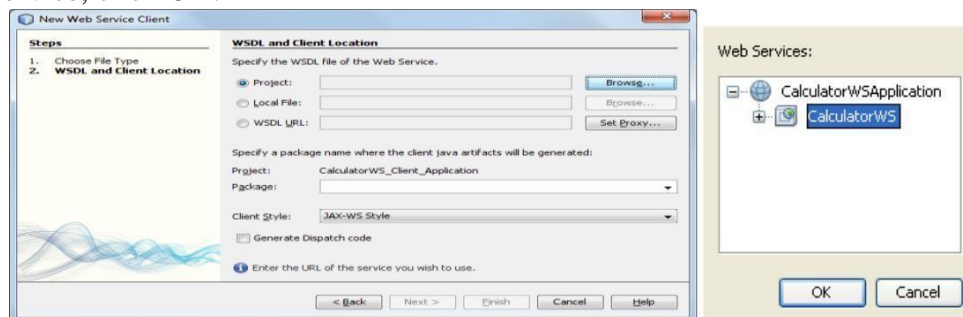
- Choose File > New Project. Select Java Application from the Java category. Name the project CalculatorWS_Client_Application. Leave Create Main Class selected and accept all other default settings. Click Finish.



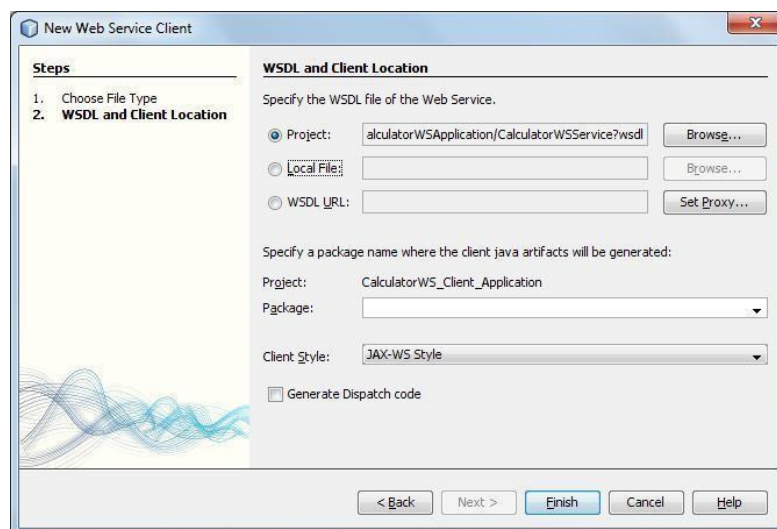
2. Right-click the CalculatorWS_Client_Application node and choose New > Web Service Client. The New Web Service Client wizard opens.



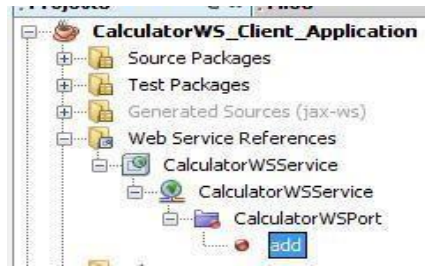
3. Select Project as the WSDL source. Click Browse. Browse to the CalculatorWS web service in the CalculatorWSApplication project. When you have selected the web service, click OK.



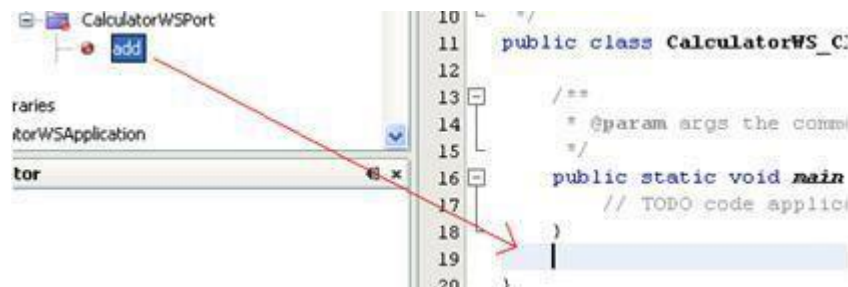
4. Do not select a package name. Leave this field empty.



5. Leave the other settings at default and click Finish. The Projects window displays the new web service client, with a node for the add method that you created:



- Double-click your main class so that it opens in the Source Editor. Drag the add node below the main() method.



You now see the following:

```
public static void main(String[] args)
{
    // TODO code application logic here
}
private static int add(int i, int j)
{
```

```
    org.me.calculator.CalculatorWS_Service service = new
    org.me.calculator.CalculatorWS_Service();
```

```
    org.me.calculator.CalculatorWS port =
    service.getCalculatorWSPort(); return port.add(i, j);
}
```

- In the main() method body, replace the TODO comment with code that initializes values for i and j, calls add(), and prints the result.

```
public static void main(String[] args)
{
    int i = 3;
    int j = 4;
    int result = add(i, j);
    System.out.println("Result = " + result);
}
```

- Surround the main() method code with a try/catch block that prints an exception.

```
public static void main(String[] args)
{
    try
    {
        int i = 3;

        int j = 4;
        int result = add(i, j);
        System.out.println("Result = " + result);
    } catch (Exception ex) {
        System.out.println("Exception: " + ex);
    }
}
```

9. Right-click the project node and choose Run.

The Output window now shows the sum:

compile:


run:

Result = 7


BUILD SUCCESSFUL (total time: 1 second)


Practical 5B: Implementing Web Service that connects to MySQL database.


Building Web Service:-

 JAX-WS is an important part of the Java EE platform.


 JAX-WS simplifies the task of developing Web services using Java technology.

 It provides support for multiple protocols such as SOAP, XML and by providing a facility for supporting additional protocols along with HTTP.

 With its support for annotations, JAX-WS simplifies Web service development and reduces the size of runtime files.

 Here basic demonstration of using IDE to develop a JAX-WS Web Service is given.

 After creating the web service, create web service clients that use the Web service over a network which is called consuming a web service.

 The client is a servlet in a web application.

❑ Let's build a Web Service that returns the book name along with its cost for a particular ISBN.

❑ To begin building this service, create the data store. The server will access the data stored in a MySQL table to serve the client.

1) Creating MySQL DB Table

create database bookshop;

use bookshop;

❑ **Create a table named Books that will store valid books information**

create table books(isbn varchar(20) primary key, bookname varchar(100), bookprice varchar(10));

❑ **Insert valid records in the Books table**

insert into books values("111-222-333","Learn My SQL","250");

insert into books values("111-222-444","Java EE 6 for Beginners","850");

insert into books values("111-222-555","Programming with Android","500");

insert into books values("111-222-666","Oracle Database for you","400");

insert into books values("111-222-777","Asp.Net for advanced programmers","1250");

2) Creating a web service

i. Choosing a container

❑ Web service can be either deployed in a Web container or in an EJB container.

❑ If a Java EE 6 application is created, use a Web container because EJBs can be placed directly in a Web application.

ii. Creating a web application

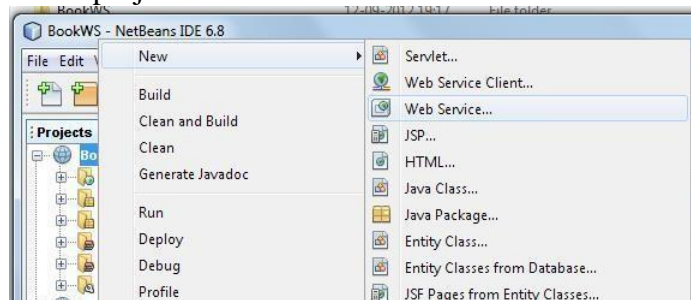
❑ To create a Web application, select File - New Project.

❑ New Project dialog box appears. Select Java Web available under the Categories section and Web Application available under the Projects section. Click Next.

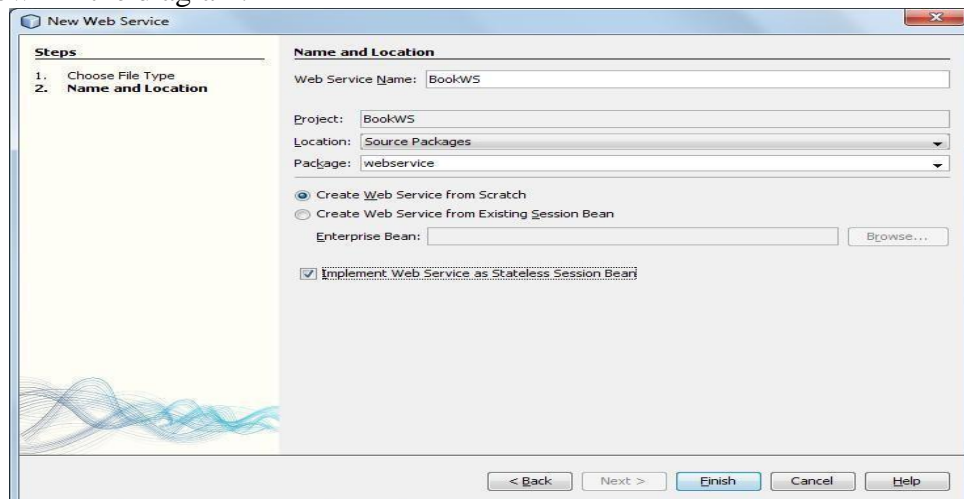
- ❑ New Web Application dialog box appears. Enter BookWS as the project name in the Project Name textbox and select the option Use Dedicated Folder for Storing Libraries.
- ❑ Click Next. Server and Settings section of the New Web Application dialog box appears. Choose the default i.e. GlassFish v3 Domain as the Web server, the Java EE 6 Web as the Java EE version and the Context Path.
- ❑ Click –Finish
- ❑ The Web application named BookWS is created.

iii. Creating a web service

- ❑ Right-click the BookWS project and select New -> Web Service as shown in diagram.



- ❑ New Web Service dialog box appears. Enter the name BookWS in the Web Service Name textbox, webservice in the Package textbox, select the option Create Web Service from scratch and also select the option implement web service as a stateless session bean as shown in the diagram.



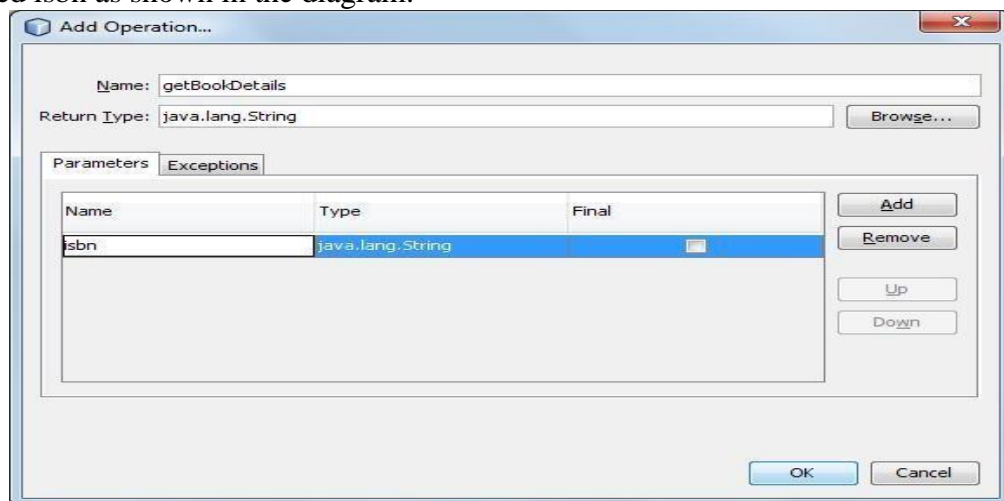
- ❑ Click Finish.
- ❑ The web service in the form of java class is ready.

3) Designing the web service

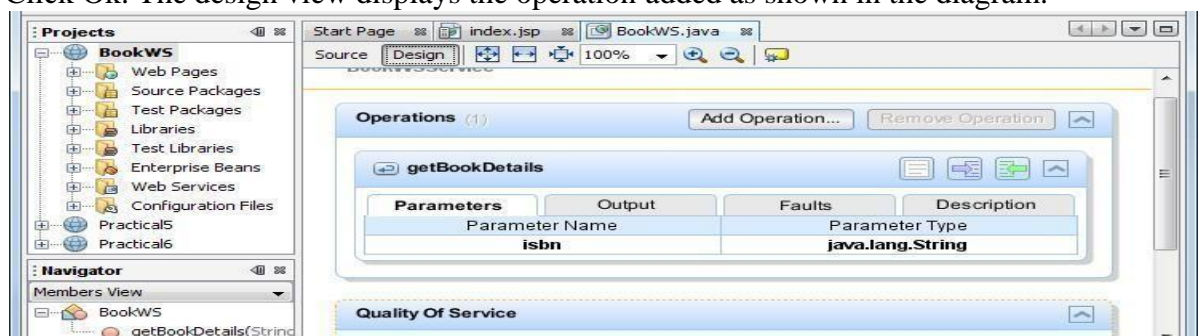
Now add an operation which will accept the ISBN number from the client to the web service.

i. Adding an operation to the web service

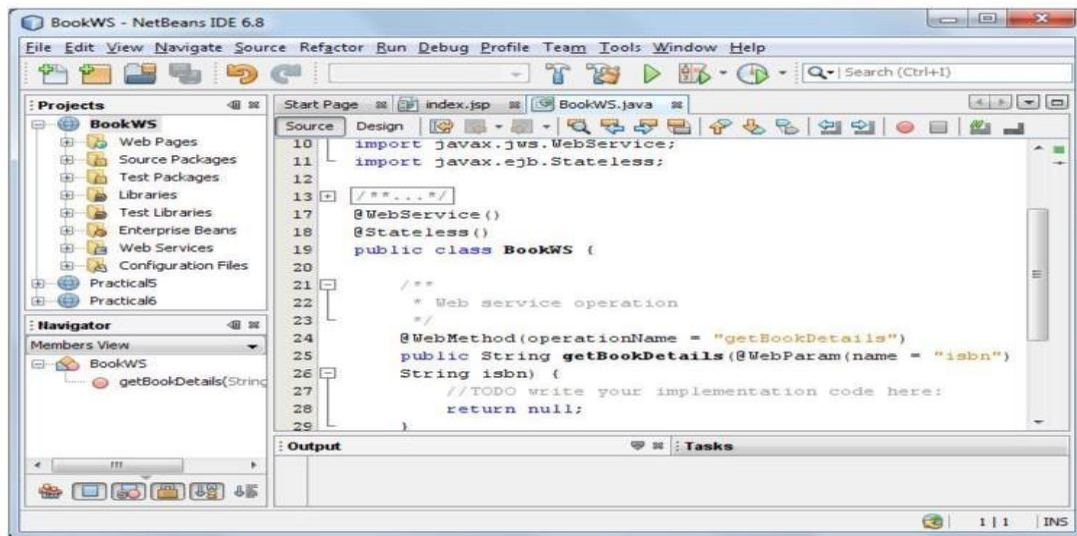
1. Change the source view of the BookWS.java to design view by clicking Design available just below the name of the BookWS.java tab.
2. The window changes as shown in the diagram.
3. Click Add Operation available in the design view of the web service.
4. Add Operation dialog appears. Enter the name getBookDetails in the Name textbox and java.lang.String in the Return Type textbox as shown in the diagram.
5. In Add Operation dialog box, click Add and create a parameter of the type String named isbn as shown in the diagram.



6. Click Ok. The design view displays the operation added as shown in the diagram.



7. Click Source. The code spec expands due to the operation added to the web service as shown in the diagram.



□ Modify the code spec of the web service BookWS.java.

Code Spec

```

package webservice;

import java.sql.*;

import javax.jws.WebMethod;

import javax.jws.WebParam;

import javax.jws.WebService;

import javax.ejb.Stateless;

@WebService()

@Stateless()

public class BookWS {

    /**

    * Web service
    operation */

    @WebMethod(operationName = "getBookDetails") public
    String getBookDetails(@WebParam(name = "isbn") String
    isbn) {

        //TODO write your implementation code here:

```

```
Connection dbcon = null;

Statement stmt = null;

ResultSet rs = null;

String query = null;

try
{
    Class.forName("com.mysql.jdbc.Driver").newInstance();

    dbcon = DriverManager.getConnection("jdbc:mysql://localhost/bookshop","root","123");

    stmt = dbcon.createStatement();

    query = "select * from books where isbn = " + isbn +
    """; rs = stmt.executeQuery(query); rs.next();

    String bookDetails = "<h1>The name of the book is <b>"
    +rs.getString("bookname") + "</b> and its cost is <b>" +rs.getString("bookprice") +
    "</b></h1>.";

    return bookDetails;
}

catch(Exception e)
{
    System.out.println("Sorry failed to connect to the database.." + e.getMessage());
}

return null;
}
}
```

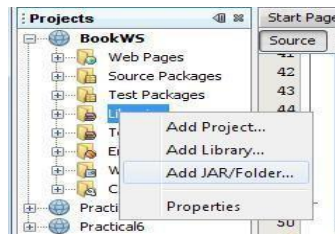
Explanation

- ❏ In the above code number entered by returned.

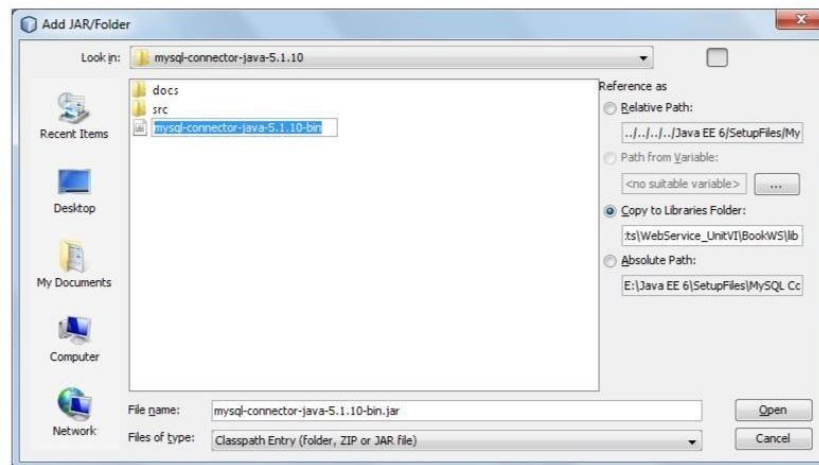
spec, a database connection is established.
Based on the ISBN the user, the associated
book name and price is retrieved and

4) Adding the MySQL connector

- ❑ We need to add a reference of MySQL connector to our web service. It is via this connector that our web service will be able to communicate with the database.
- ❑ Right click on the libraries and select Add JAR/Folder as shown in the diagram.



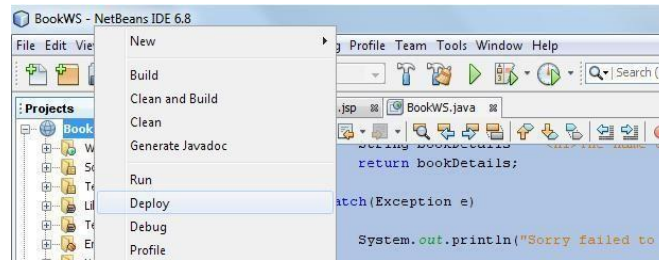
- ❑ Choose the location where mysql-connector-java-5.1.10-bin is located, select it and click on open as shown.



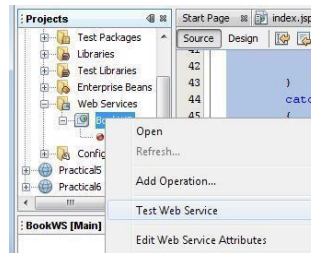
5) Deploying and testing the web service

- ❑ When a web service is deployed to a web container, the IDE allows testing the web service to see if it functions as expected.

- ❑ The tester application provided by GlassFish, is integrated into the IDE for this purpose as it allows the developer to enter values and test them.
- ❑ No facility for testing whether an EJB module is deployed successfully is currently available.
- ❑ To test the BookWS application, right click the BookWS project and select Deploy as shown in the diagram.



- ❑ The IDE starts the server, builds the application and deploys the application to the server.
- ❑ Follow the progress of these operations in the BookWS (run-deploy) and GlassFish v3 Domain tabs in the Output view.
- ❑ Now expand the web services directory of the BookWS project, right-click the BookWS Web service and select Test web service as shown in the diagram.



- ❑ The IDE opens the tester page in the web browser, if the web application is deployed using GlassFish server as shown in the figure.



- ❑ Enter the ISBN number as shown in the diagram.
- ❑ Click getBookDetails. The book name and its cost are displayed as shown in the diagram.

← → ↻ localhost:35637/BookWS/BookWSService?Tester

getBookDetails Method invocation

Method parameter(s)

Type	Value
java.lang.String	111-222-333

Method returned

java.lang.String : "The name of the book is Learn My SQL and its cost is 250<,"

SOAP Request

```
<?xml version="1.0" encoding="UTF-8"?>
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
  <S:Header/>
  <S:Body>
    <ns2:getBookDetails xmlns:ns2="http://webservice/">
      <isbn>111-222-333</isbn>
    </ns2:getBookDetails>
  </S:Body>
</S:Envelope>
```

SOAP Response

```
<?xml version="1.0" encoding="UTF-8"?>
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
```

6) Consuming the web service

- ☐ Once the web service is deployed, the next most logical step is to create a client to make use of the web service's getBookDetails() method.

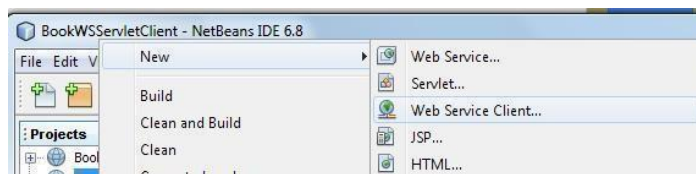
i. Creating a web application

- ☐ To create a web application, select File -> New Project.
- ☐ New project dialog box appears, select java web available under the categories section and web application available under the projects section. Click Finish.
- ☐ New web application dialog box appears. Enter BookWSServletClient as the project name in the Project Name textbox and select the option Use Dedicated Folder for Storing Libraries.

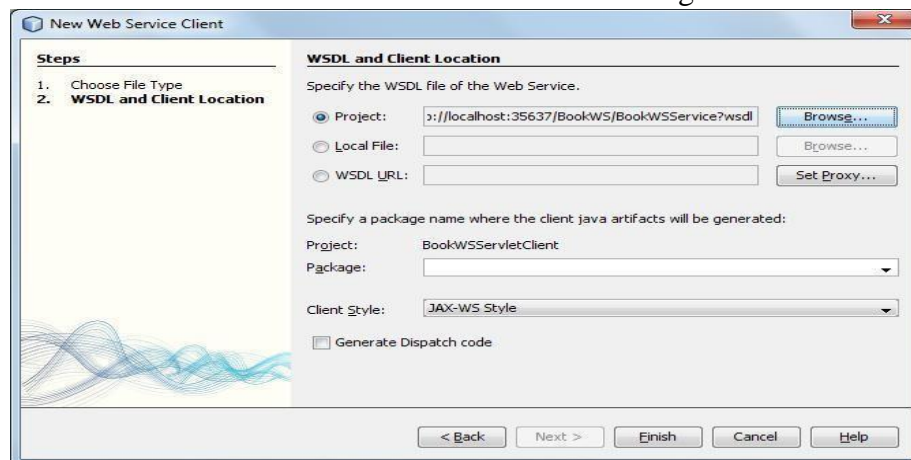
- ❑ Click Next. Server and settings section of the new web application, dialog box appears. Choose the default i.e. GlassFish v3 Domain as the web server, the Java EE 6 web as the Java EE version and the context path.
- ❑ Click Finish.
- ❑ The web application named BookWSServletClient is created.

ii. Adding the web service to the client application

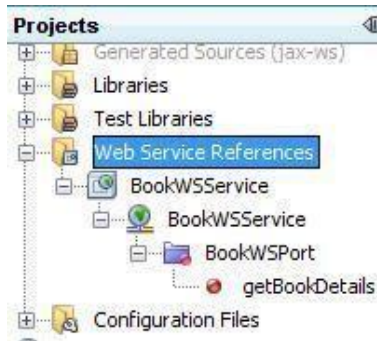
- ❑ Right-click the BookWSServletClient project and select New -> Web Service Client as shown in the diagram.



- ❑ New Web Service Client dialog box appears. In the Project section, click Browse and browse through the web service which needs to be consumed. Click ok. The name of the web service appears in the New Web Service Client as shown in the diagram.



- ❑ Leave settings as it is. Click Finish
- ❑ The Web Service Reference directory is added to the BookWSServletClient application as shown in the diagram. It displays the structure of the newly created client including the getBookDetails() method created earlier.



iii. Creating a servlet

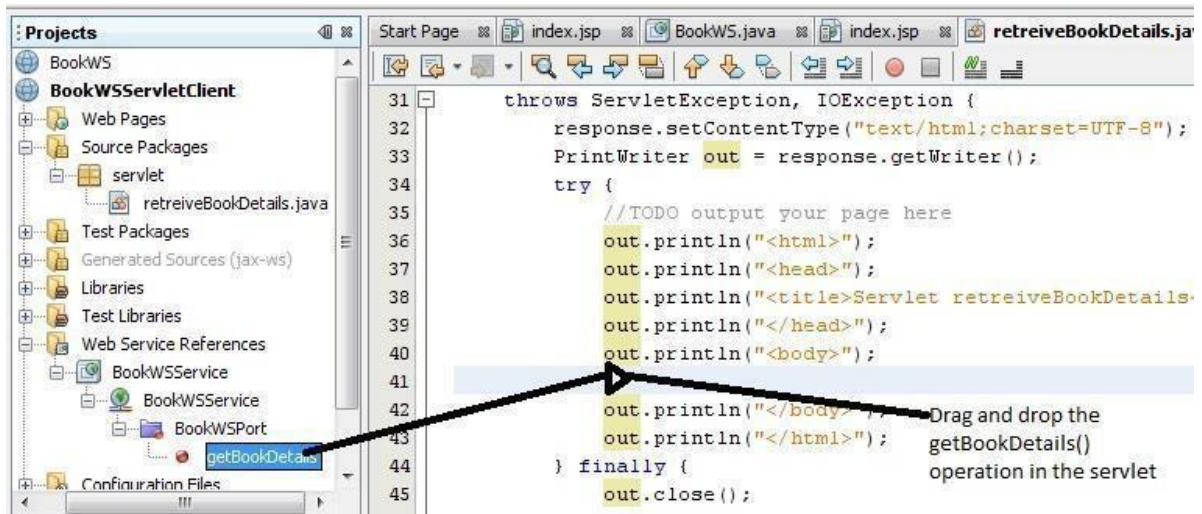
- ❑ Create retrieveBookDetails.java using NetBeans IDE.
- ❑ Right click source package directory, select New -> Servlet.
- ❑ New Servlet dialog box appears. Enter retrieveBookDetails in the Class Name textbox and enter servlet in the package textbox.
- ❑ Click Next. Configure Servlet Deployment section of the New Servlet dialog box appears. Keep the defaults.
- ❑ Click Finish.
- ❑ This creates the servlet named retrieveBookDetails.java in the servlet package.
- ❑ retrieveBookDetails.java is available with the default skeleton created by the NetBeans IDE which needs to be modified to hold the application logic.
- ❑ In the retrieveBookDetails.java source code, remove the following comments available in the body of the processRequest() method.

`/*TODO output your page here*/`

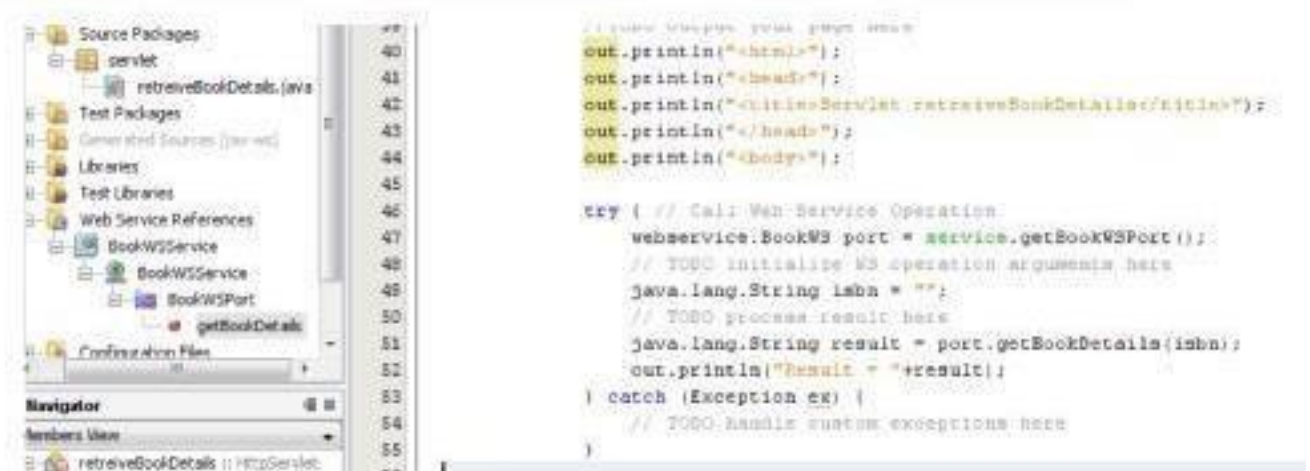
- ❑ Replace the following code spec:

```
out.println("<h1>Servlet retrieveBookDetails at " + request.getContextPath () + "</h1>");
```

With the code spec of the getBookDetails() operation of the web service by dragging and dropping the getBookDetails operation as shown in the diagram.



❑ The Servlet code spec changes as shown in the diagram



❑ The web service is instantiated by the @WebServiceRef annotation.

❑ Now change the following code spec:

```
java.lang.String isbn = "";
```

to

```
java.lang.String isbn = request.getParameter("isbn");
```

iv. Creating an HTML form

- ❑ Once the web service is added and the servlet is created, the form to accept ISBN from the user needs to be coded.
- ❑ Since NetBeans IDE by default [as a part of Web Application creation] makes available index.jsp file. Modify it to hold the following code spec.

```
<% @page contentType="text/html" pageEncoding="UTF-8"%> <!DOCTYPE
HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"

"http://www.w3.org/TR/html4/loose.dtd">

<html>

<head>

<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>SOAP Cleint - Get Book Details</title>

</head>

<body bgcolor="pink">

<form name="frmgetBookDetails" method="post" action="retreiveBookDetails">
<h1>

ISBN : <input type="text" name="isbn"/><br><br>
</h1>

<input type="submit" value="Submit"/>
</form>

</body>

</html>
```

v. Building the Web Application

- ❑ Build the web application.
- ❑ Right click BookWSServletClient project and select Build.

- Once the Build menu item is clicked the details about the compilation and building of the BookWSServletClient Web application appears in the output – BookWSServletClient (dist) window.

vi. Running the Application

- Once the compilation and building of the web application is done run the application.
- Right click the BookWSServerCleint project and select run.
- Once the run processing completes in NetBeans IDE a web browser is automatically launched and the BookWSServletCleint application is executed as shown in the diagram.
- Enter the ISBN as shown in the diagram



- Click Submit. The book name and its cost are displayed as shown in the diagram.

