Web Services

Q.) What is distributed technology?

- A technology is said to be distributed if it's business objects are geographically dispersed (across multiple JVM's) and still communicating one another.

Q.) Why distributed technology?

- Load balancing
- High Availability
- High Processing

Q.) What are different distributed technologies?

- There are so many distributed technologies
  - CORBA
  - RMI
  - EJB
  - WEB SERVICES ...etc.

CORBA

- More complexity in the implementation
- Heavy weight
- Language dependent
- Commercial
- Plat from independent

RMI

- Lack of Enterprise services/Middle ware services
- Open source
- No acknowledgement
- Language dependent
- Plat from independent

EJB

- Supports Enterprise services/Middle ware services in the declarative manner
- Having multiple flavors of APIs to handle different requirements
  - Session beans
    - Stateless
    - Stateful
  - Entity beans(not using now)
    - JPA
  - Message Driven Beans

- Managed by Application server
- Robust
- Heavy weight
- Language Dependent
- Plat from independent

WEB SERVICES

- Supports Enterprise services
- Interoperable (Language independent & Plat from independent)

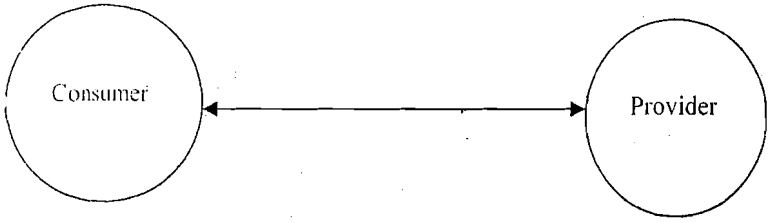
So, Web services is interoperable distributed technology

Q.) What we require if two persons need to communicate?

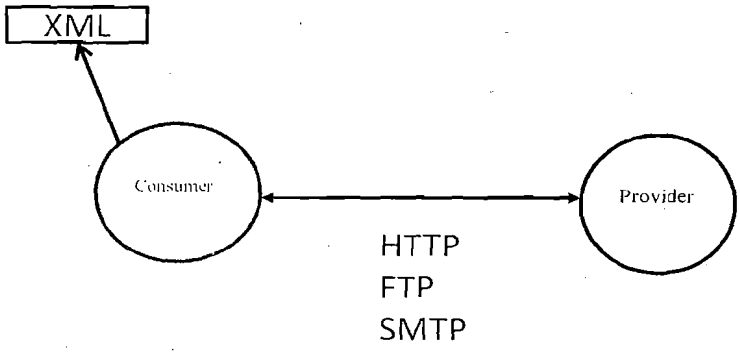
1.) Two persons ☺
2.) Medium
3.) Language
4.) Rules(protocol)

Web services architecture

If two applications needs to communicate



- Both the applications should be connected (network)
- Both application needs to communicate by following some rules(protocols)
  - HTTP
  - FTP
  - SMTP...etc.
- To communicate language is required(Neutral language)
  - XML

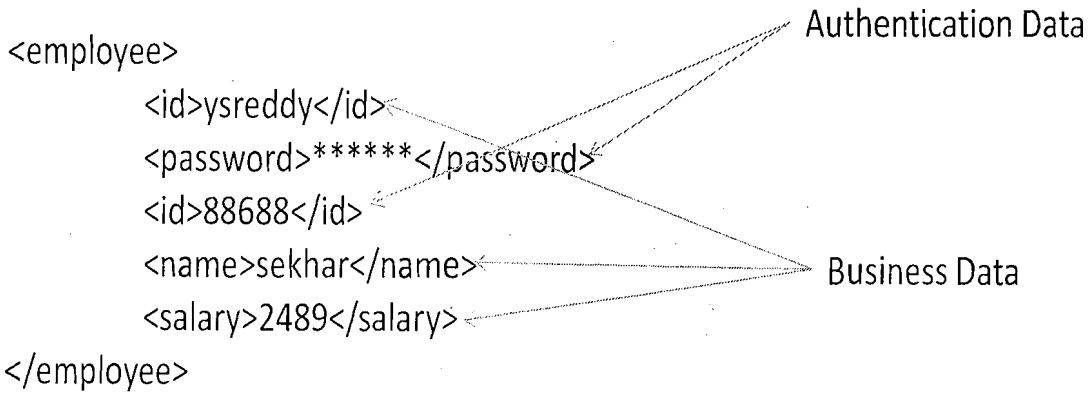


- In the internet world we prefer to use HTTP protocol, and we have so many technologies which supports HTTP protocol.
- Lets assume that we are sending employee information form "Consumer" to "Provider" in the XML format as follows.

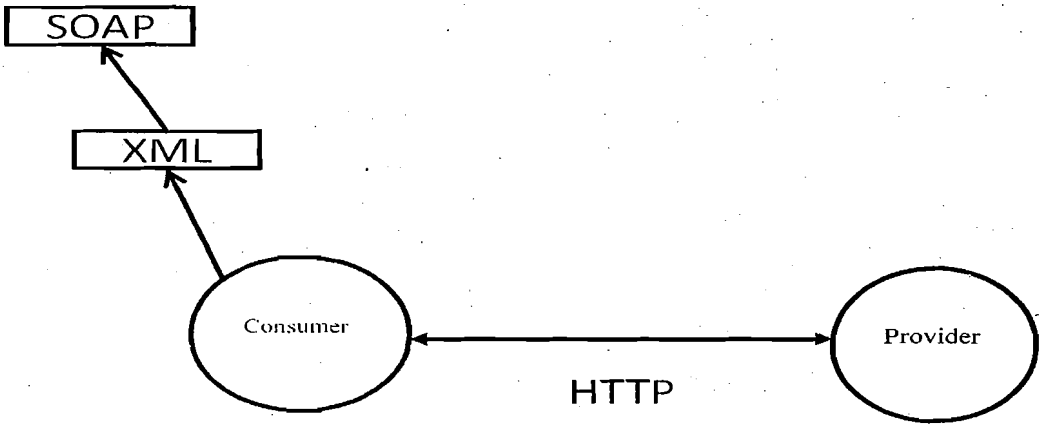
```
<employee>
        <id>ysreddy</id>
        <password>******</password>
        <id>88688</id>
        <name>sekhar</name>
        <salary>2489</salary>
</employee>
```

```
<employee>
        <id>ysreddy</id>          <-- Authentication Data
        <password>******</password>
        <id>88688</id>            <-- Business Data
        <name>sekhar</name>
        <salary>2489</salary>
</employee>
```

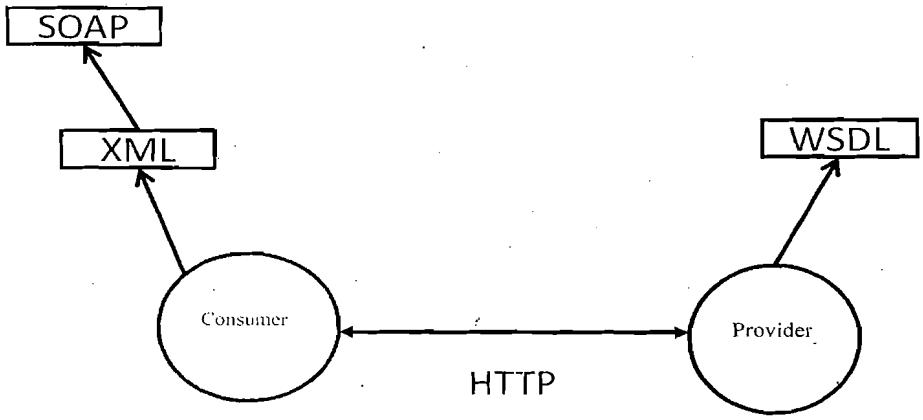
- At "Provider" side it may be interpreted as follows



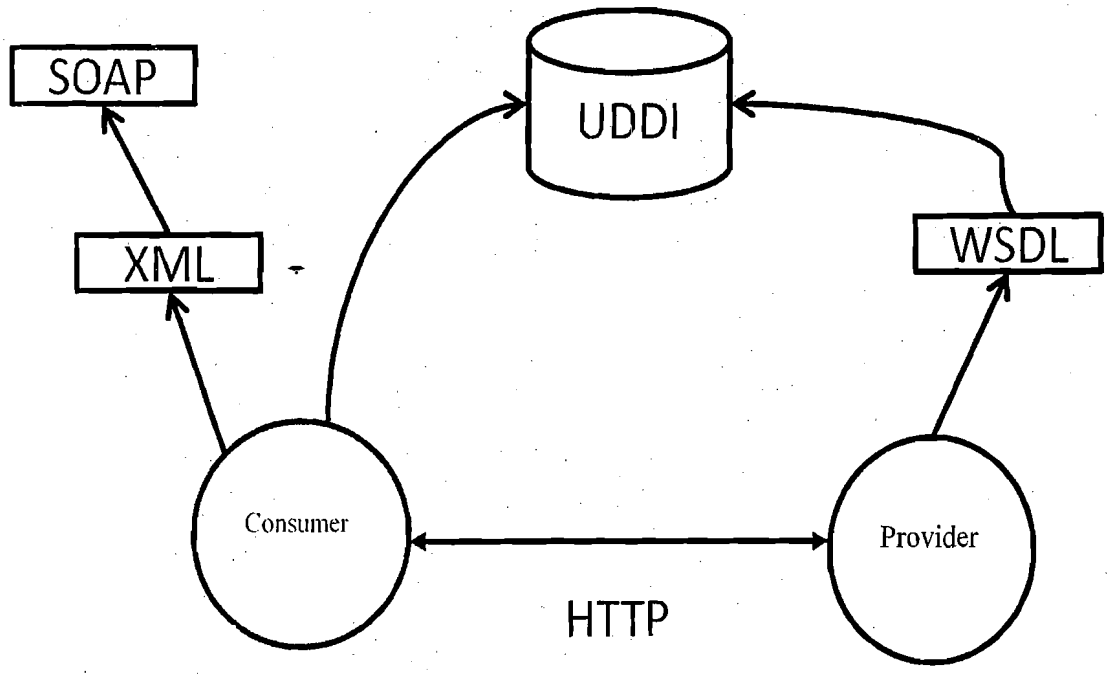
- So, classification of the information is missing.
- So, we don't send raw XML document(contains business data) to the providers.
- We need classification of the request data like
  - Processing information/Authentication information
  - Business information
- To achieve this we will go for **SOAP**
- SOAP stands for **Simple Object Access Protocol**
- SOAP is a classification protocol
- SOAP is a binding protocol(processing data + business data)

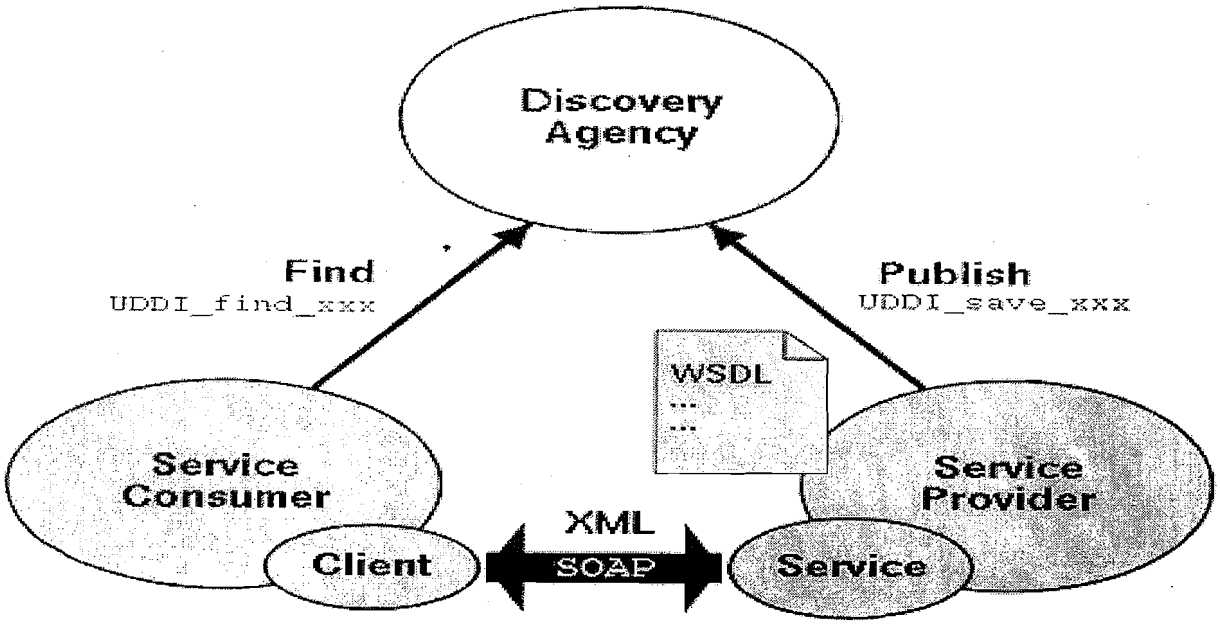


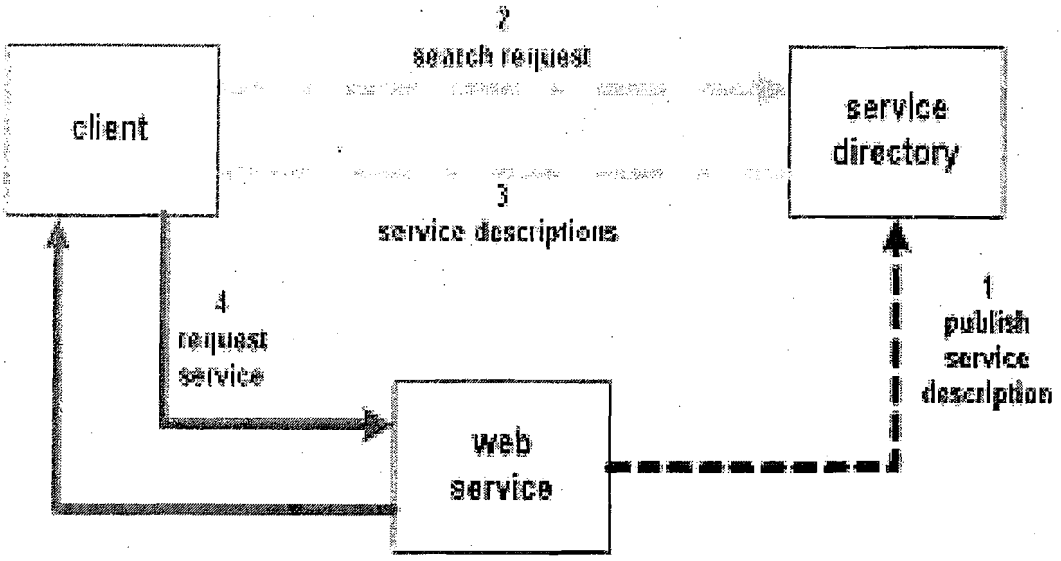
- But what services are providing by the Provider should be explained. To know the services of "Provider", "Consumer" cannot look into the code of of "Provider".
- So "Consumer" needs the information like what are the services are providing, what they will take as input, and what the output they will return, what url has to call to get services...etc.
- "Provider" will explain all these information in one document called "**WSDL**".
- **"WSDL"** stands for **Web Service Description Language**
- **"WSDL"** is an XML document, because it should be language independent, so that any type of client can understand the services of the "Provider".



- **"WSDL"** is the document which explains the services information. But how can "Consumer" get that **WSDL** document, from where "Consumer" get?
- So **"WSDL"** documents has to be placed in some location from where "Consumer" can access them, that location is nothing but **"UDDI"**
- UDDI stands for **Universal Description Discovery and Integration**
- UDDI is the Registry where all the **WSDL** documents are registered
- UDDI should be interoperable, so it is also developed in XML
- UDDI registry also called XML registry







2.) What is WS-I?

- The WS-I (Web Services Interoperability) Organization is an association of IT industry companies, including IBM and Microsoft, that aim to create Web services specifications that all companies can use.

2.) What are the different standards given by WS-I to implement Web services ?

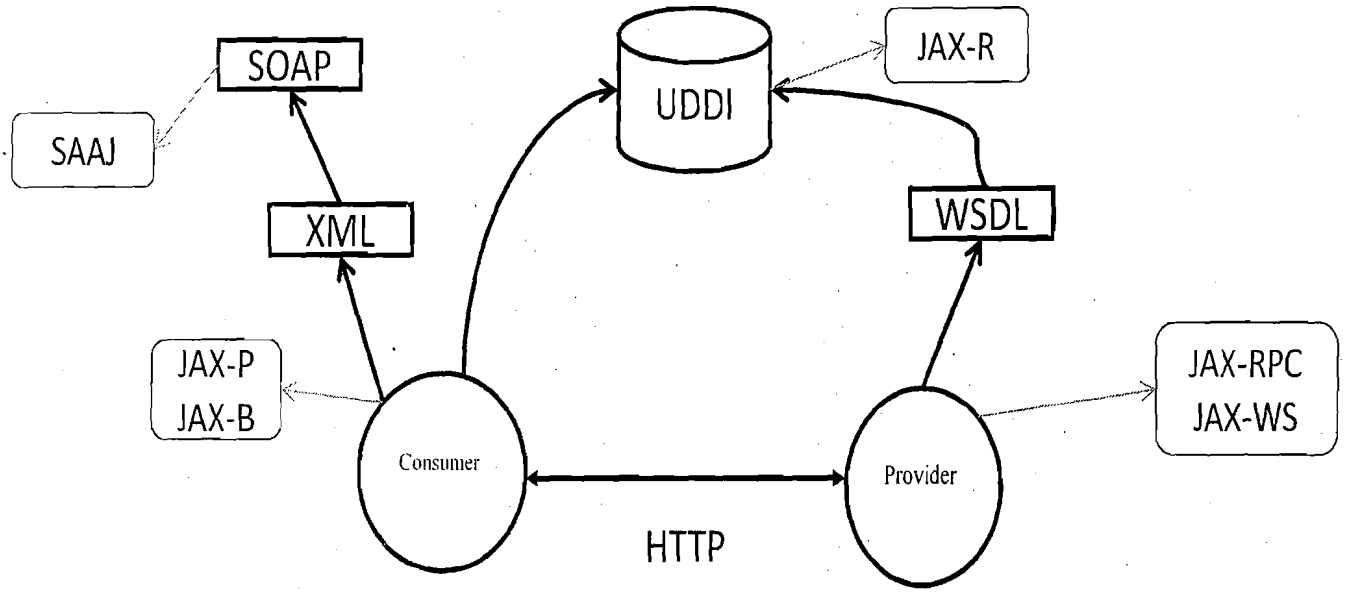
- WS-I has given
  - Basic Profile 1.0 (BP 1.0)
  - Basic Profile 1.1 (BP 1.1)
  - Basic Profile 2.0 (BP 2.0)

Q.) What are Web services?

- Web services is a software application that confirms the WS-I's given speicifiations(BP 1.0/BP 1.1/BP 2.0) is said to be web services.
- Any program that is distributed in nature, language independent and plat form neutral is said to be web services.

There so many API's, Implementations for the WS-I given standards

- Sun has released "JAX-RPC" API that follows BP 1.0 standard
- Sun has released "JAX-WS" API that follows BP 1.1 standard



JAX-RPC : Java APIs for XML-based Remote Procedure Call

JAX-WS : Java API for XML Web Services

JAX-P : Java API for XML Processing

JAX-B : Java Architecture for XML Binding

JAX-R : Java API for XML Registries

SAAJ : SOAP with Attachments API for Java

Specification (from WS-I)	API (from Sun microsystem)	Implementation (from different companies)

BP 1.0	JAX-RPC	JAX-RPC-SI(from sun) Apache-Axis(from ASF)...etc
BP 1.1	JAX-WS	JAX-WS RI(from Sun) Apache-Axis2(from ASF) Metro(from Sun) CXF(from ASF)...etc.

NOTE: JAX-WS RI(Reference Implementation) was failed to access by .Net, means failed in interoperability. Sun then released "metro" implementation of JAX-WS, which resolved interoperability problem and also added some more features.

NOTE: Apache CXF internally uses spring.

Web Services Development

Web services development can done in two ways

1. Contract first(top-down) approach
2. Contract last(bottom-up) approach

Contract first Approach

WSDL → Services

Contract last Approach

Services → WSDL

NOTE:

- Services can be developed with any technology like .Net, Java, Php...etc
- In java, Services can be developed using any API like JAX-RPC, JAX-WS, RESTFUL...etc..
- In java, Services can be developed using any implementations like sun, apache ...etc..
- But WSDL development is unique, it's not plat form specific, not language specific, not technology specific, not API specific, not implementation specific.

So, we can say we are developing web services using

- JAX-RPC, Contract first approach
- JAX-RPC , Contract last approach

NOTE: In the internet world we prefer to use HTTP protocol, and we have so many technologies which supports HTTP protocol.

Q.) What are the HTTP protocols implemented API's?

1. Servlet API
2. EJB API

Q.) What is end point?

- A component which receives "Consumer" request
- In general we use either servlet or ejb as an End Point.

So, we can say we are developing web services using

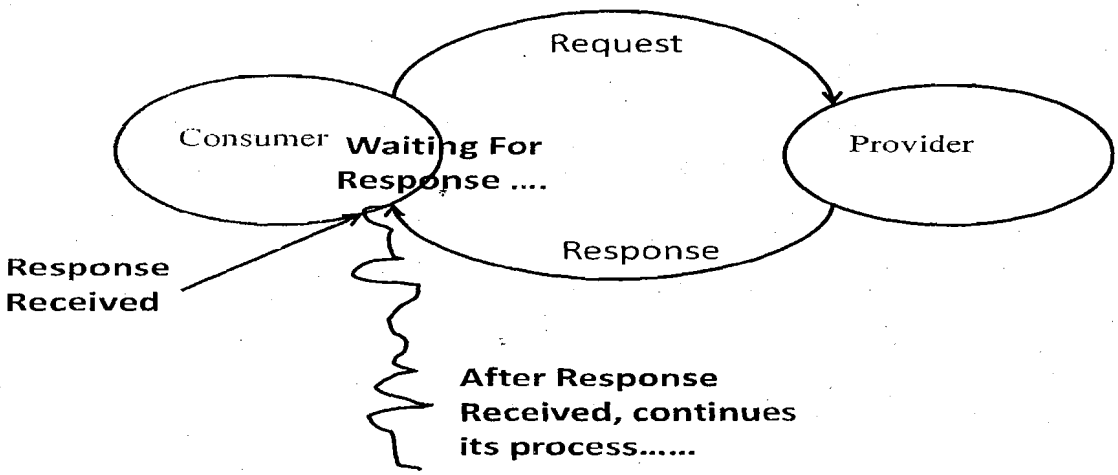
- JAX-RPC, Servlet end point url, Contract first approach
- JAX-RPC, EJB end point url, Contract last approach

MEP

- MEP stands for Message Exchanging Patterns
- "Consumer" can communicate with "Provider" in three ways
  - Synchronous request-reply
  - Asynchronous request-reply
  - Fire and forget

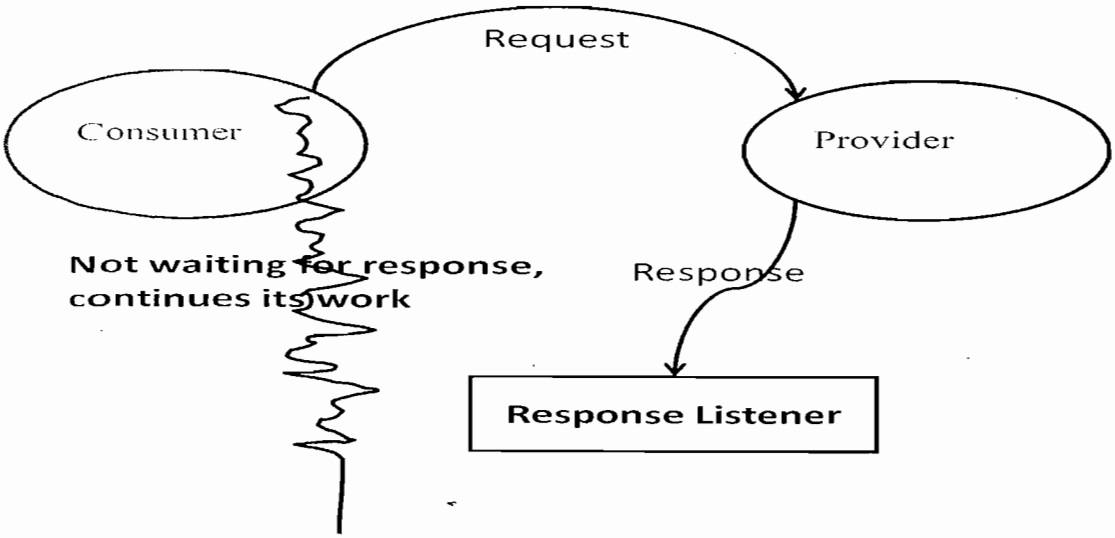
Synchronous request-reply

- In this "Consumer" sends the request to "Provider", and "Consumer" blocks until response comes back from the "Provider", Until response comes back "Consumer" can't proceed.



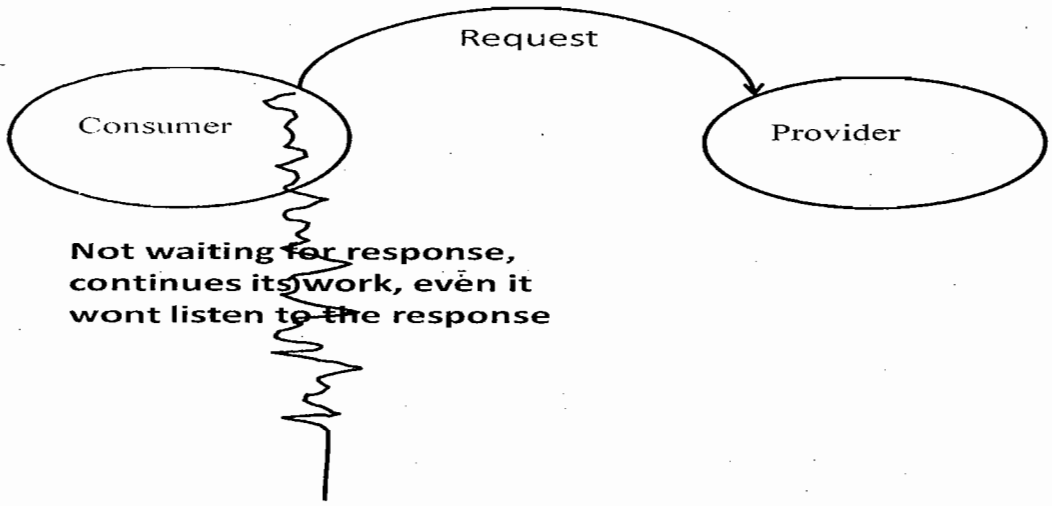
Asynchronous request-reply

- In this "Consumer" sends the request to "Provider", and "Consumer" will not wait until it gets the response.
- After request sent to the "Provider" it continues its flow of execution
- "Consumer" will have "**Response Listener**" which listens the response from the "Provider".



Fire and forget

- In this "Consumer" sends the request to "Provider", and "Consumer" will not wait until it gets the response. Even it won't have any Response listener
- Once the request is sent to the "Provider" it continues its process, doesn't bather about response.



So, we can say we are developing web services using

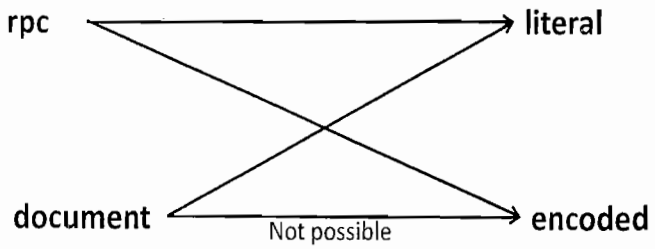
- JAX-RPC, Servlet end point url, Contract first approach, synchronous request-reply
- JAX-RPC, Servlet end point url, Contract first approach, asynchronous request-reply
- JAX-RPC, Servlet end point url, Contract first approach, fire and forget

MEF

- MEF stands for **Message Exchanging Formats**
- When "Consumer", "Provider" exchanging the information, that information can happens in the following formats.
  - document-literal
  - rpc-literal
  - rpc-encoded

- Default MEF is "rpc-encoded"

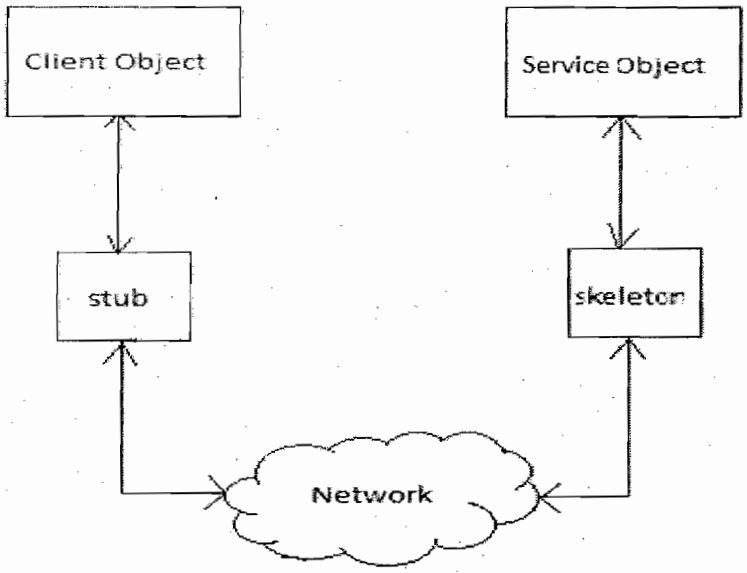
literal → As its is XML is exchanging
encoded → UTF-8, UTF-16



So, we can say we are developing web services using

- JAX-RPC, Servlet end point url, Contract first approach, synchronous request-reply, rpc-encoded
- JAX-RPC, Servlet end point url, Contract first approach, synchronous request-reply, rpc-literal
- JAX-RPC, Servlet end point url, Contract first approach, synchronous request-reply, document-literal

Q.) How the Provider and Consumer communicate in distribute technology?



Server object side requirement

1.) Interface for the service object

2.) Service object

3.) Skeleton

Client side requirement

1.) Interface for the service object

2.) Stub

Q.)What is a stub? What are its functions?

- A stub is a client side proxy for the service object
- Stub is a java object that implements the same interface which the service object implements
- Stub is client socket program
- Stub is generated by a tools

Functions of Stub

- Client makes a method call on stub
- Performing marshalling of the input arguments
- Passing the method call along with marshaled input arguments to the skeleton
- Receiving the result from the skeleton
- Unmarshalling the result returned by the skeleton
- Returning unmarshalled result to the client

Q.) What is a skeleton? What are its responsibilities?

- Skeleton is a server side proxy for the service object
- Skeleton is a server socket program
- Skeleton is generated by tools

Functions of Skeleton

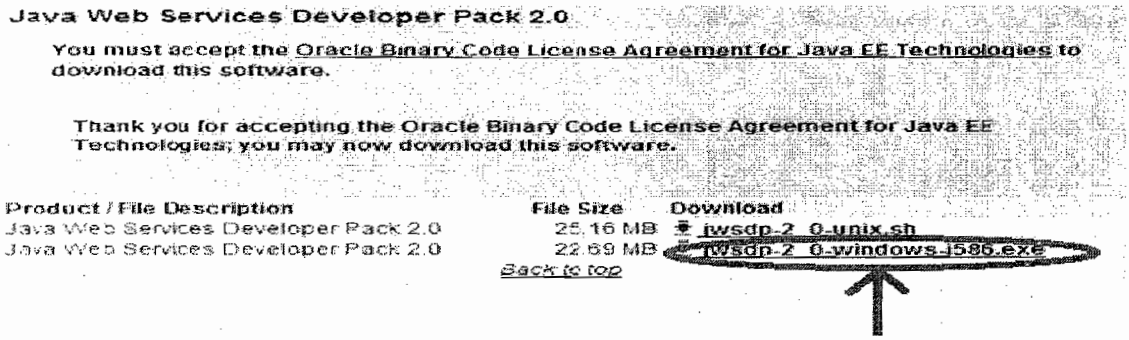
- Receiving the method call from the stub
- Unmarshalling the method input arguments
- Invoking actual method call on the service object
- Receiving the result from the service object
- Marshalling the result
- Returning the marshalled results to the stub

Web services development with JAX-RPC

1. Down load jwsdp-2_0-windows-i586.exe from the bellow URL

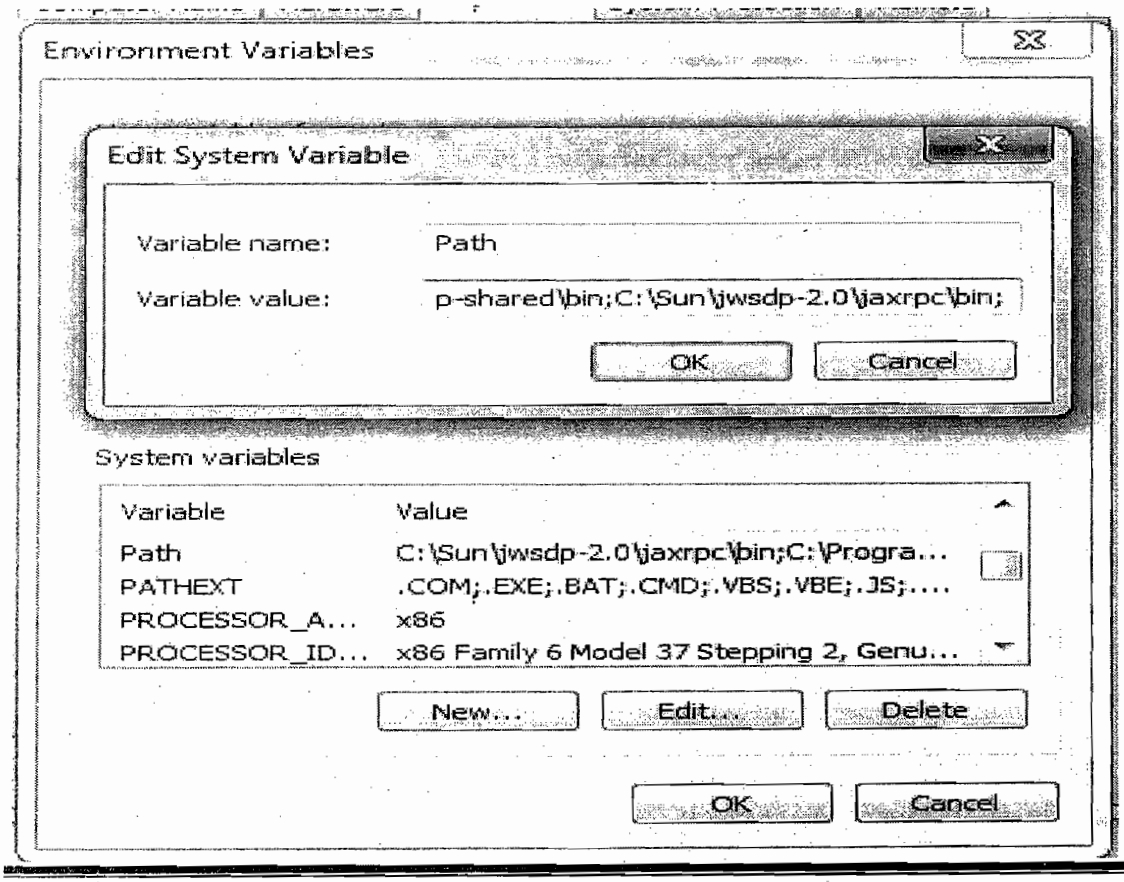
http://www.oracle.com/technetwork/java/javasebusiness/downloads/java-archive-downloads-jwsdp-419428.html#jwsdp-2.0-oth-JPR

- Java Web Services Developer Pack 1.5 Plug-in
- JavaServer Web Development Kit 1.0.1
- JavaServer Web Development Kit 1.0



The file `jwsdp-2_0-windows-i586.exe` is the Java WSDP installer. After downloading is finished double click on the installer icon and click on Run button to install.

After the installation, we need to set the path to jax-rpc.



Now check by typing **wscompile –help** in command prompt it should a long list of options.

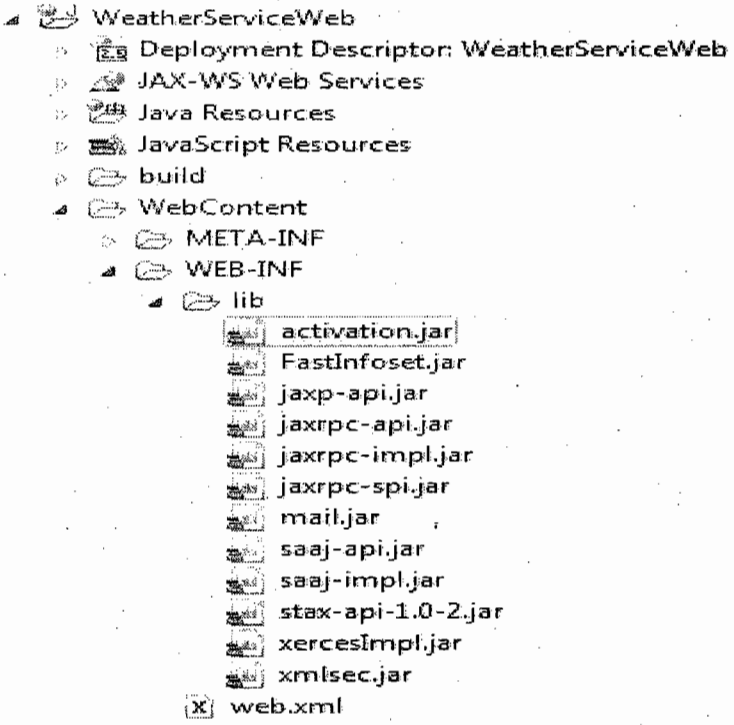
```
C:\Users\welcome>wscompile -help

Usage: wscompile [options] configuration_file

where [options] include:
  -classpath <path>          specify where to find input class files
  -cp <path>                 same as -classpath <path>
  -d <directory>             specify where to place generated output files
  -define                    define a service
  -f:<features>              enable the given features (see below)
  -features:<features>       same as -f:<features>
  -g                         generate debugging info
  -gen                       same as -gen:client
  -gen:client                generate client artifacts (stubs, etc.)
  -gen:server                generate server artifacts (ties, etc.)
  -help                      display help
  -source <version>          generate code for the specified JAXRPC SI version.
                             supported versions are: 1.0.1, 1.0.3, 1.1, 1.1.1 and
ault)
  -httpproxy:<host>:<port>   specify a HTTP proxy server (port defaults to 8080)
  -import                    generate interfaces and value types only
  -keep                      keep generated files
  -model <file>              write the internal model to the given file
  -nd <directory>            specify where to place non-class generated files
  -o                         optimize generated code
```

2. Eclipse Project

- In order to build web service, first start your eclipse and create a new **Dynamic Web Project** for example with name **WeatherServiceWeb**. While creating your project make sure you configure the **target runtime**(configure tomcat in eclipse). Here in our examples we are using Tomcat 6.0, you can use any J2EE compliant Servlet container for this example.

- After setting up the project copy all the required jar bundles in to project **WebContent/WEBINF/lib** directory.

- Below picture depicts your project structure.



3. Java End

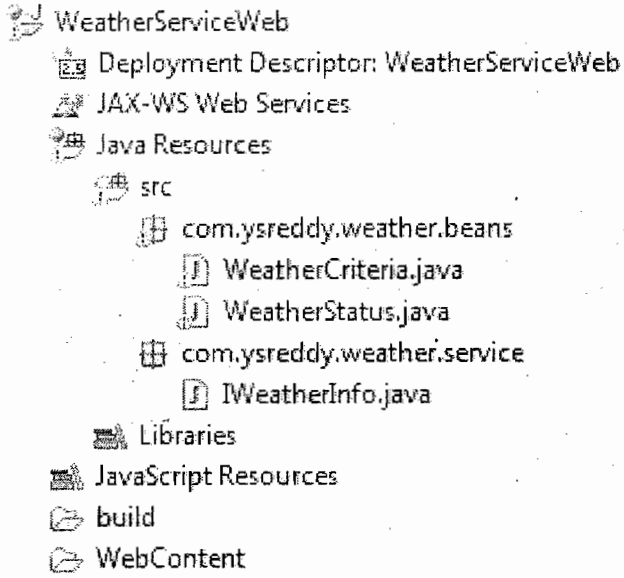
- Now create a new package e.g. **com.ysreddy.weather.beans** under which create two java beans which acts as a request and response objects for your web service e.g. **WeatherCriteria** and **WeatherStatus** respectively.

- Your request and response objects should abide to the rules of Standard Java Beans convention and your objects should be **serializable**.

- Create one more new package e.g. **com.ysreddy.weather.service** under which, create a **new Service Endpoint Interface** which acts as a contract between consumer and provider.

- Following are the rules you should keep in mind while creating a SEI interface
  1. Our Service Endpoint Interface should extend from **java.rmi.Remote** Interface
  2. Declare all business methods that we want to expose as webservice methods.
  3. All the methods that are delcared in SEI interface should be public and should have parameters and return values in serializable in nature.
  4. All our Web Service Methods should be declared to throw **RemoteException**.

- Following the above rules lets build a SEI interface which looks as below.



WeatherCriteria.java

```
1. package com.ysreddy.weather.beans;
2.
3. import java.io.Serializable;
4.
5. public class WeatherCriteria implements Serializable {
6.     private String city;
7.     private String state;
8.     private String country;
9.
10.         public String getCity() {
11.               return city;
12.         }
```

```
13.
14.         public void setCity(String city) {
15.               this.city = city;
16.         }
17.
18.         public String getState() {
19.               return state;
20.         }
21.
22.         public void setState(String state) {
23.               this.state = state;
24.         }
25.
26.         public String getCountry() {
27.               return country;
28.         }
29.
30.         public void setCountry(String country) {
31.               this.country = country;
32.         }
33.
34.   }
```

WeatherStatus.java

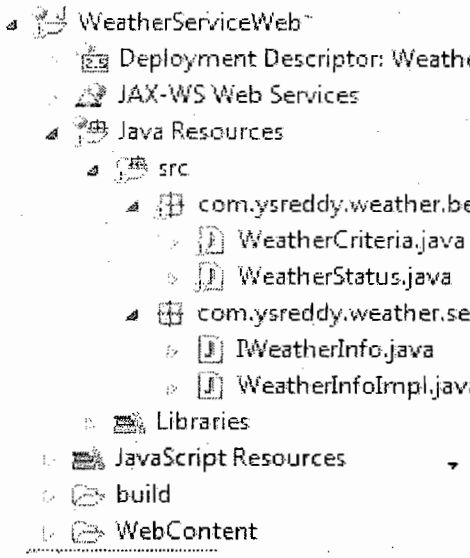
```
1. package com.ysreddy.weather.beans;
2.
3. import java.io.Serializable;
4.
5. public class WeatherStatus implements Serializable {
6.     private int temperature;
7.     private double humidity;
8.     private String description;
9.
10.         public int getTemperature() {
11.               return temperature;
12.         }
13.
14.         public void setTemperature(int temperature) {
15.               this.temperature = temperature;
16.         }
17.
18.         public double getHumidity() {
19.               return humidity;
20.         }
21.
22.         public void setHumidity(double humidity) {
23.               this.humidity = humidity;
24.         }
25.
26.         public String getDescription() {
27.               return description;
28.         }
29.
30.         public void setDescription(String description) {
31.               this.description = description;
```

```
32.         }
33.
34.    }
```

IWeatherInfo.java

```
1. package com.ysreddy.weather.service;
2.
3. import java.rmi.Remote;
4. import java.rmi.RemoteException;
5.
6. import com.ysreddy.weather.beans.WeatherCriteria;
7. import com.ysreddy.weather.beans.WeatherStatus;
8.
9. public interface IWeatherInfo extends Remote {
10.         public WeatherStatus getWeatherInfo(WeatherCriteria weatherCriteria)
11.                                                                 throws RemoteException;
12.    }
```

➪ Now create a Java Class under the same package, which implements the above defined SEIinterface and override all the methods declared in your SEI interface. Implement your business logic in your Web Service Methods as shown below.



WeatherInfoImpl.java

```
1. package com.ysreddy.weather.service;
2.
3. import com.ysreddy.weather.beans.WeatherCriteria;
4. import com.ysreddy.weather.beans.WeatherStatus;
5.
6. public class WeatherInfoImpl implements IWeatherInfo {
7.    public WeatherStatus getWeatherInfo(WeatherCriteria weatherCriteria){
8.
9.          if (weatherCriteria != null) {
10.                   System.out.println("city :" + weatherCriteria.getCity());
11.                   System.out.println("state :" + weatherCriteria.getState());
12.                   System.out.println("country :" + weatherCriteria.getCountry());
13.             }
```

```
14.             WeatherStatus status = new WeatherStatus();
15.             status.setTemperature(101);
16.             status.setHumidity(12.5);
17.             status.setDescription("Today may be rained or may not be");
18.             return status;
19.         }
20.   }
```

Note:- web service methods in the implementation class are not required to throw **RemoteException**.

4. Configuration

- Once you are finished with your java part. Now its time to write couple of XML files to generate Stub/Skeletons and binding information which performs the handshaking process.

- Create a new **config.xml** configuration file under WEB-INF directory, which contains the information that describe our web service. A sample config.xml is as follows.

config.xml

```
1. <?xml version="1.0" encoding="UTF-8"?>
2.    <!--
3.          Copyright 2004 Sun Microsystems, Inc. All rights reserved. SUN
4.          PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
5.    -->
6. <configuration xmlns="http://java.sun.com/xml/ns/jax-rpc/ri/config">
7.    <service name="WeatherService"
8.          targetNamespace="http://ysreddy.com/weather/wsdl"
9.          typeNamespace="http://ysreddy.com/weather/types"
10.               packageName="com.ysreddy.weather.service.binding">
11.               <interface name="com.ysreddy.weather.service.IWeatherInfo"
12.                     servantName="com.ysreddy.weather.service.WeatherInfoImpl" />
13.         </service>
14.   </configuration>
```

- In the configuration file you should modify the **targetNameSpace**, **typeNameSpace**, **PackageName** under which you want to generate binding classes.

- In the **<interface>** subelement **name** attribute specifies the service endpoint interface and **serventName** attribute specifies the class that implements endpoint interface.

NOTE: The **<configuration>** element may contain exactly one **<service> or <wsdl> or <modelfile>** element.

5. wscompile

- The wscompile tool generates stubs, ties, serializers and WSDL files used in JAX-RPC clients and services. The tool reads a configuration file, which specifies either WSDL file, a compiled service endpoint interface.

Syntax: wscompile [options] <configuration-file>

Option	Description
-classpath <path>	specify where to find input class files
-cp <path>	same as -classpath <path>
-d <directory>	specify where to place generated output files
-gen:both	generate both client and server artifacts
-gen:client	generate client artifacts (stubs, etc.)
-gen:server	generate server artifacts (ties, etc.) and the WSDL file (If you are using wsdeploy you do not specify this option.)
-keep	keep generated files
-model <file>	write the internal model to the given file
-verbose	output messages about what the compiler is doing

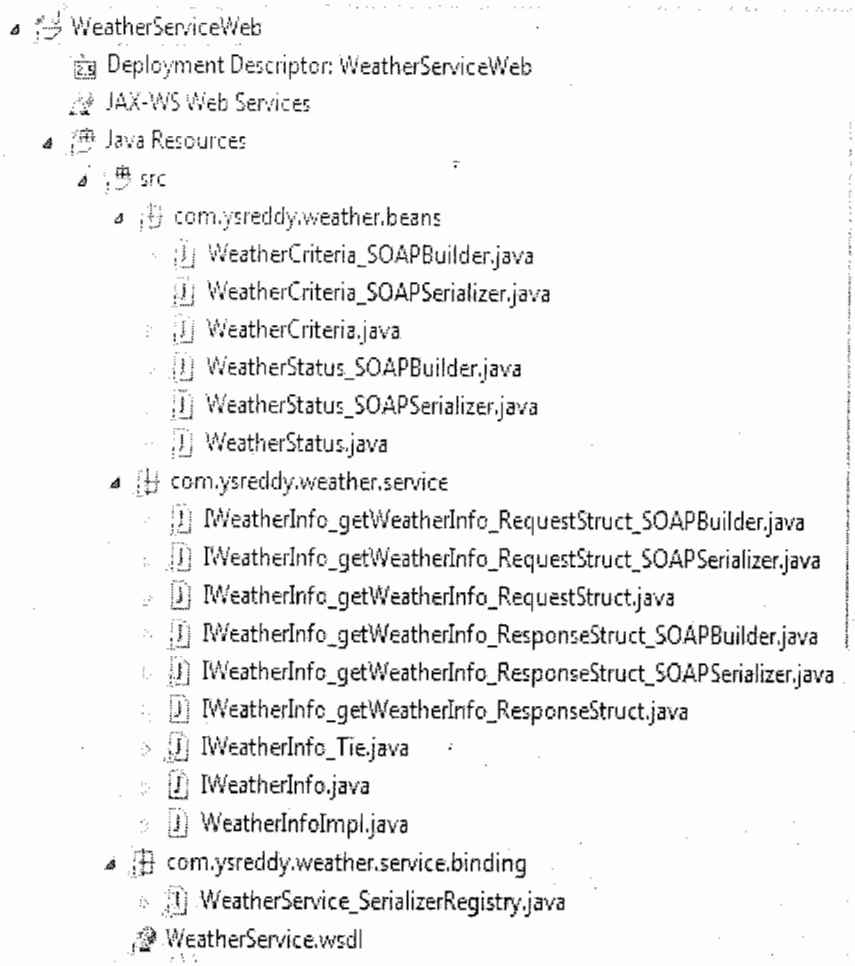
NOTE: Note that exactly one of the –import or –define or –gen options should be used.

➪ Now navigate to the root directory of our project in the command prompt and execute the following command.

E:\ws\WeatherServiceWeb>**wscompile -gen:server -cp build/classes -d src -keep -model model-rpc-enc.xml.gz -verbose WebContent\WEB-INF\config.xml**

```
C:\Windows\system32\cmd.exe
E:\ws\WeatherServiceWeb>wscompile -gen:server -cp build/classes -d src -keep -model model-
rpc-enc.xml.gz -verbose WebContent\WEB-INF\config.xml
[creating model: WeatherService]
[creating service: WeatherService]
[creating port: com.ysreddy.weather.service.IWeatherInfo]
[creating operation: getWeatherInfo]
[CustomClassGenerator: generating JavaClass for: getWeatherInfo]
[CustomClassGenerator: generating JavaClass for: getWeatherInfoResponse]
[SOAPObjectSerializerGenerator: writing  serializer/deserializer for: {http://ysreddy.com/
weather/types}getWeatherInfo]
[SOAPObjectSerializerGenerator: writing  serializer/deserializer for: {http://ysreddy.com/
weather/types}WeatherCriteria]
[SOAPObjectSerializerGenerator: writing  serializer/deserializer for: {http://ysreddy.com/
weather/types}getWeatherInfoResponse]
[SOAPObjectSerializerGenerator: writing  serializer/deserializer for: {http://ysreddy.com/
weather/types}WeatherStatus]
[SOAPObjectBuilderGenerator: writing object builder for: getWeatherInfo]
[SOAPObjectBuilderGenerator: writing object builder for: WeatherCriteria]
[SOAPObjectBuilderGenerator: writing object builder for: getWeatherInfoResponse]
[SOAPObjectBuilderGenerator: writing object builder for: WeatherStatus]
[SerializerRegistryGenerator: creating serializer registry: com.ysreddy.weather.service.bi
nding.WeatherService_SerializerRegistry]
E:\ws\WeatherServiceWeb>
```

⇨ Now go to your project and refresh it, should start showing up the generated stubs/skeleton files as shown below.



⇨ This command along with generating binding classes, it will also generates a model file and wsdl file under source.

Optional

{

- If we want our own url pattern for our web service we can give that in the WSDL document as follows

```
<service name="WeatherService">
        <port name="IWeatherInfoPort" binding="tns:IWeatherInfoBinding">
                <soap:address location="http://localhost:8086/WeatherServiceWeb/getWeather" />
        </port>
</service>
```

}

⇨ Now drag and drop them into **WEB-INF** directory. Now create one more configuration file **jaxrpc-ri.xml** file under **WEB-INF** directory.

jaxrpc-ri.xml

```
1. <?xml version="1.0" encoding="UTF-8"?>
2.
3. <!--
4.  Copyright 2004 Sun Microsystems, Inc. All rights reserved.
5.  SUN PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
6. -->
7.
8. <webServices
9.         xmlns="http://java.sun.com/xml/ns/jax-rpc/ri/dd"
10.        version="1.0"
11.        targetNamespaceBase="http://ysreddy.com/weather/wsdl"
12.        typeNamespaceBase="http://ysreddy.com/weather/types"
13.        urlPatternBase="/ws">
14.
15.        <endpoint
16.            name="IWeatherInfo"
17.            displayName="Weather Web Service"
18.            description="Get Weather Web Service"
19.            wsdl="/WEB-INF/WeatherService.wsdl"
20.            interface="com.ysreddy.weather.service.IWeatherInfo"
21.            implementation="com.ysreddy.weather.service.WeatherInfoImpl"
22.            model="/WEB-INF/model-rpc-enc.xml.gz"/>
23.
24.        <endpointMapping
25.            endpointName="Weather"
26.            urlPattern="/getWeather"/>
27.
28.    </webServices>
```

- **<webservices>** element must contain one or more **<endpoint>** elements. In this example, note that the interface and implementation attributes of **<endpoint>** specifies the service's interface and implementation class. The **<endpointMapping>** element associates the service name with a URL pattern.

- Now export your project as **war** onto your desktop by selecting from file menu and choosing export project.

6. wsdeploy

- **wsdeploy** tool reads a WAR file and then generates another WAR file that is capable for deployment. Basically this tool updates content in web.xml file.
- Now navigate to the directory where we exported our project and execute the following command.

C:\Users\welcome\Desktop>**wsdeploy -verbose -o target.war WeatherServiceWeb.war**

```
C:\Users\welcome\Desktop>wsdeploy -verbose -o target.war WeatherServiceWeb.war
info: created temporary directory: C:\Users\welcome\AppData\Local\Temp\jaxrpc-deploy-9c679
2
[CustomClassGenerator: Class com.ysreddy.weather.service.IWeatherInfo_getWeatherInfo_Reque
stStruct exists. Not overriding.]
[CustomClassGenerator: Class com.ysreddy.weather.service.IWeatherInfo_getWeatherInfo_Respo
nseStruct exists. Not overriding.]
[SOAPObjectSerializerGenerator: writing serializer/deserializer for: {http://ysreddy.com/
weather/types}getWeatherInfo]
[SOAPObjectSerializerGenerator: writing serializer/deserializer for: {http://ysreddy.com/
weather/types}WeatherCriteria]
[SOAPObjectSerializerGenerator: writing serializer/deserializer for: {http://ysreddy.com/
weather/types}getWeatherInfoResponse]
[SOAPObjectSerializerGenerator: writing serializer/deserializer for: {http://ysreddy.com/
weather/types}WeatherStatus]
[SOAPObjectBuilderGenerator: writing object builder for: getWeatherInfo]
[SOAPObjectBuilderGenerator: writing object builder for: WeatherCriteria]
[SOAPObjectBuilderGenerator: writing object builder for: getWeatherInfoResponse]
[SOAPObjectBuilderGenerator: writing object builder for: WeatherStatus]
[SerializerRegistryGenerator: Class com.ysreddy.weather.service.binding.WeatherService_Ser
ializerRegistry exists. Not overriding.]
[TieGenerator: Class com.ysreddy.weather.service.IWeatherInfo_Tie exists. Not overriding.]

info: created output war file: C:\Users\welcome\Desktop\target.war
```

- Now unzip the **targer.war**, you should be able to **find jaxrpc-ri-runtime.xml** and **web.xml**.
- Copy these both files into **WEB-INF** directory. Deploy the project and start sever.

You can find the following changes in web.xml

web.xml

```
1. <?xml version="1.0" encoding="UTF-8"?>
2. <web-app xmlns="http://java.sun.com/xml/ns/javaee"
3.     xmlns:web="http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
4.     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" id="WebApp_ID"
5.     version="2.5"
6.     xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
7.      http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd">
8.     <listener>
9.         <listener-class>com.sun.xml.rpc.server.http.JAXRPCContextListener
10.                                                          </listener-class>
11.    </listener>
12.    <servlet>
13.        <servlet-name>IWeatherInfo</servlet-name>
14.        <servlet-class>com.sun.xml.rpc.server.http.JAXRPCServlet</servlet-class>
15.        <load-on-startup>1</load-on-startup>
```

```
16.    </servlet>
17.    <servlet-mapping>
18.          <servlet-name>IWeatherInfo</servlet-name>
19.          <url-pattern>/ws/IWeatherInfo</url-pattern>
20.    </servlet-mapping>
21.
22.    </web-app>
```

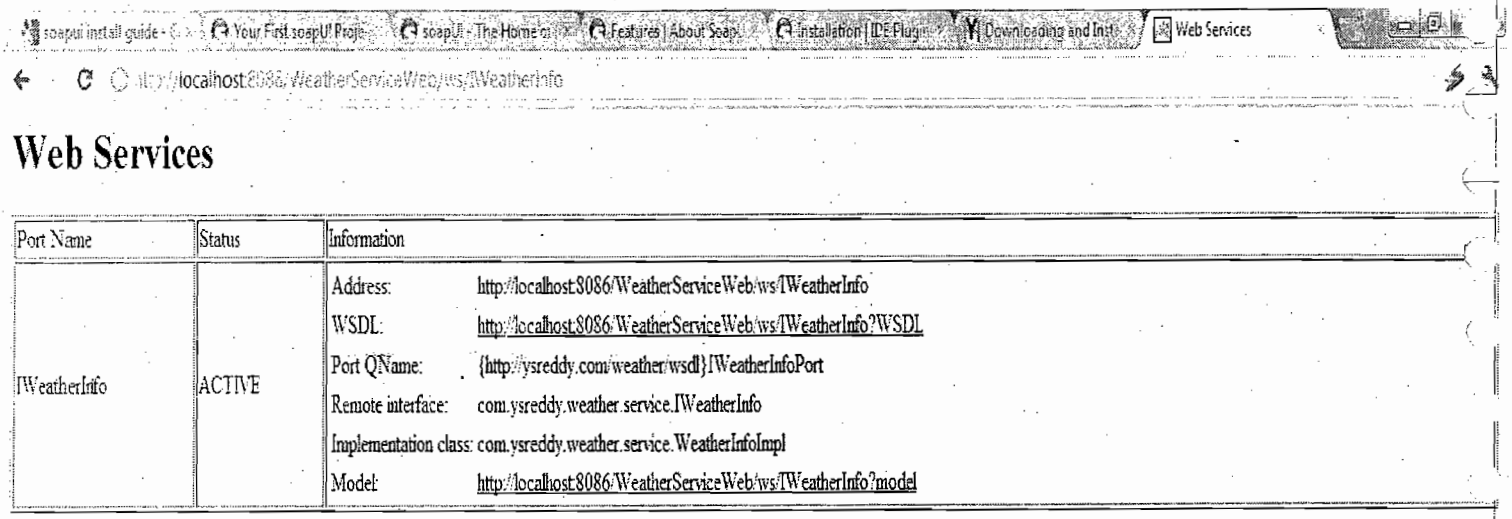
NOTE: Open jaxrpc-ri-runtime.xml file and make sure that tie="com.ysreddy.weather.service.IWeatherInfo_Tie" is available.

⇨ Now deploy your project into servlet container like tomcat and start server.

⇨ Now access your webservice wsdl with the below url format in a web browser.
http://<domain>:<port>/<context-root>/<servletpath/url pattern of your service>?wsdl

In our example type the following url

http://localhost:8086/WeatherServiceWeb/ws/IWeatherInfo

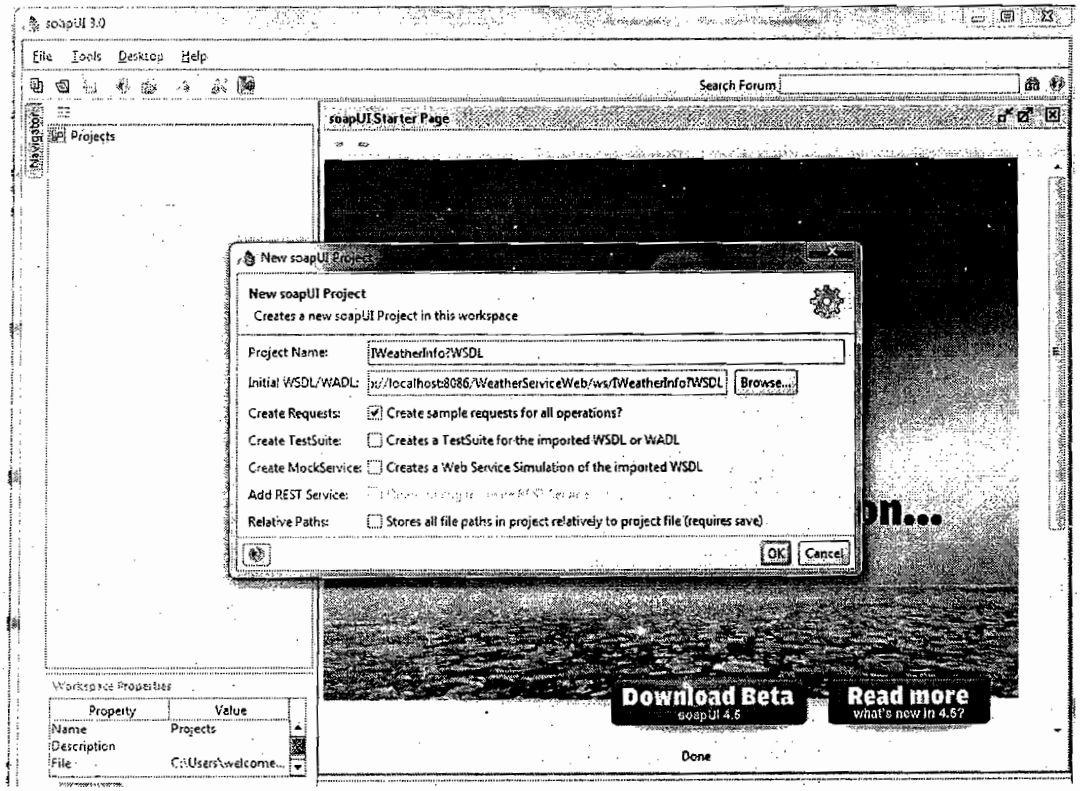


⇨ Now click on WSDL hyper link, then WSDL file will be displayed.

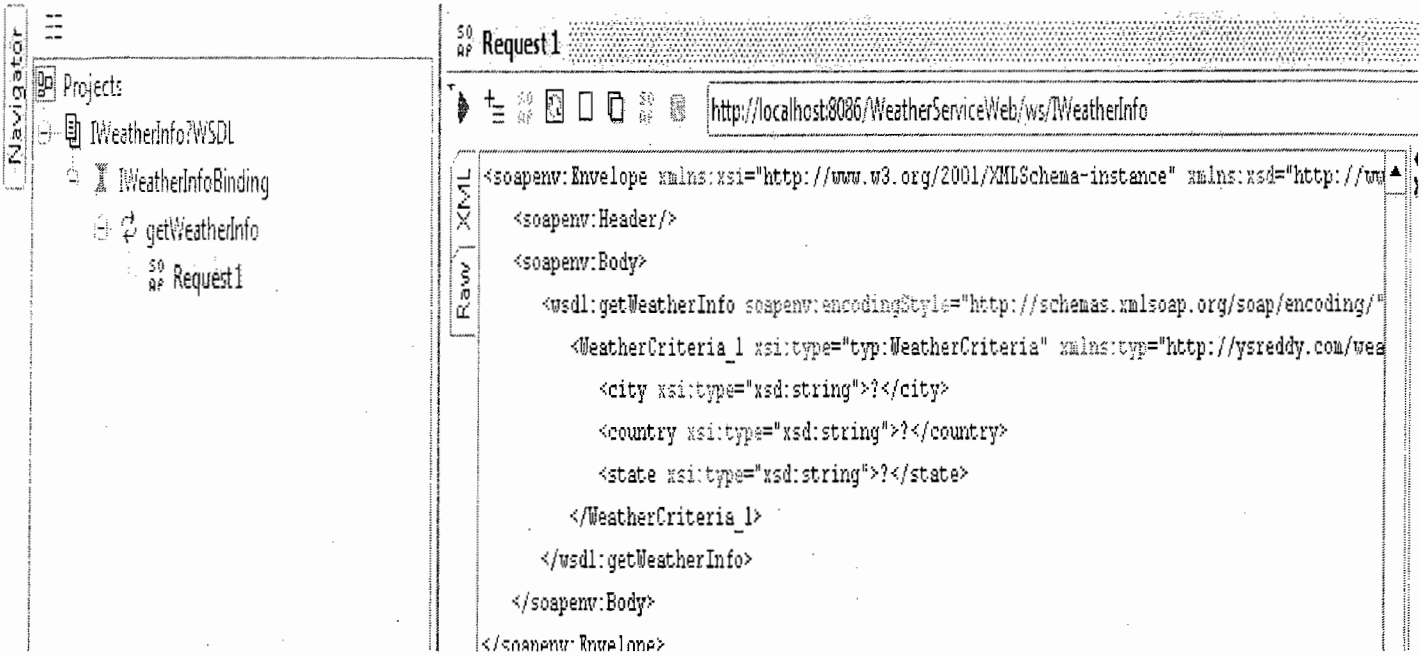
⇨ **Now Open SoapUI, by double click on Installed Icon**

⇨ **Now Right click on "Projects" → "New SoapUI Project"**

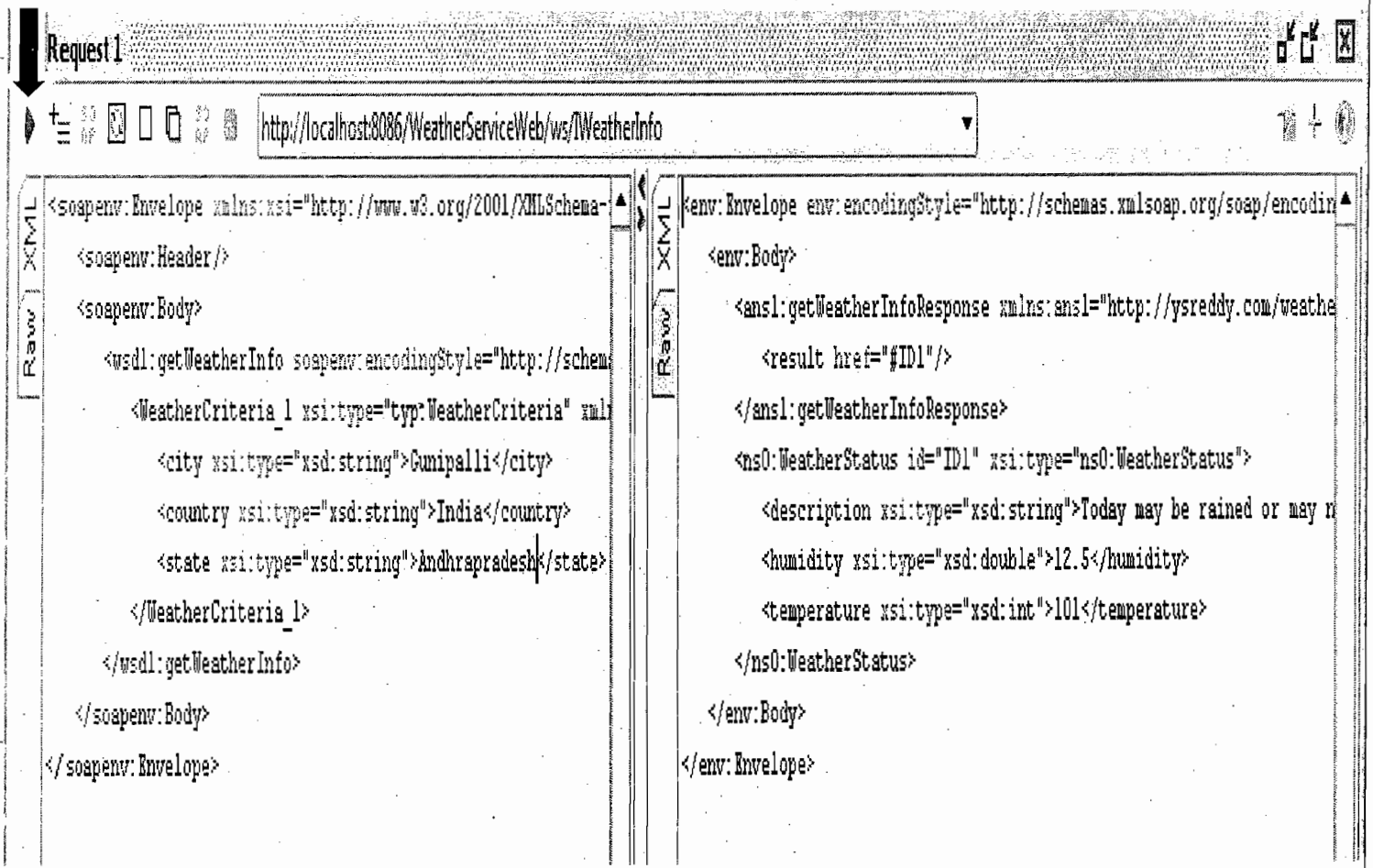
➪ **Now paste our WSDL location(http://localhost:8086/WeatherServiceWeb/ws/IWeatherInfo?WSDL) in "Initial WSDL/WADAL" And click on "OK"**



➪ **Now expand the project as follows, and double click on "Request 1"**



➪ **Now give input values and click on "Run" Button, Then we will get the response as shown below**



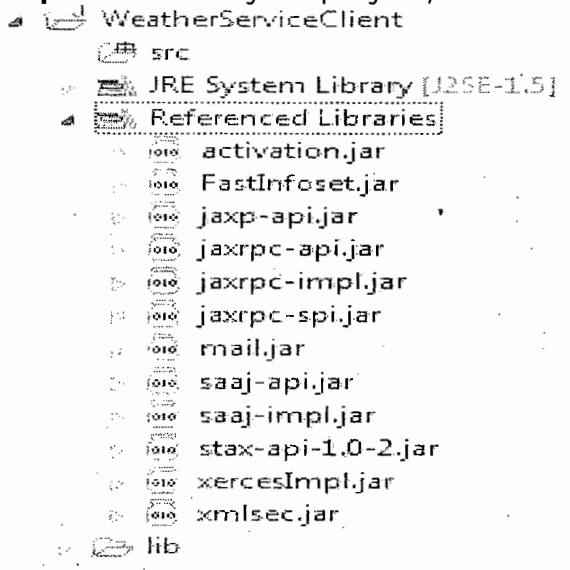
JAX-RPC Web service client

We can develop Web service client using JAX-RPC in three ways

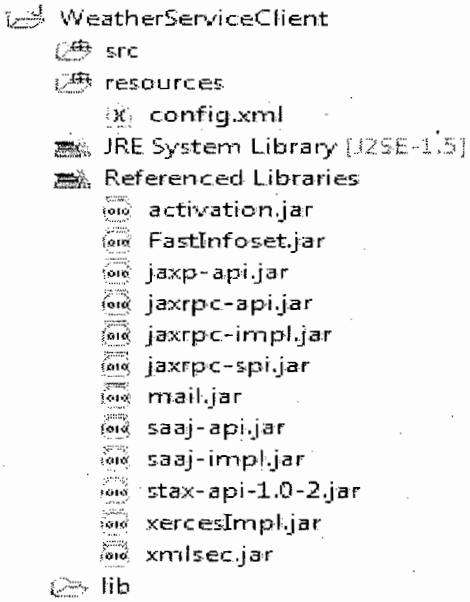
1. Stub-based
   - ⇨ By creating stubs at client side
2. Dynamic-proxy based
   - ⇨ Without stubs at client side
3. Dynamic-Invocation-Interface
   - ⇨ Our own serializer, deserializer

1.Stub-based client

Step 1: Create a java project, and add the required jars



Step 2: Create a source folder with name "resources" and create a config.xml as follows



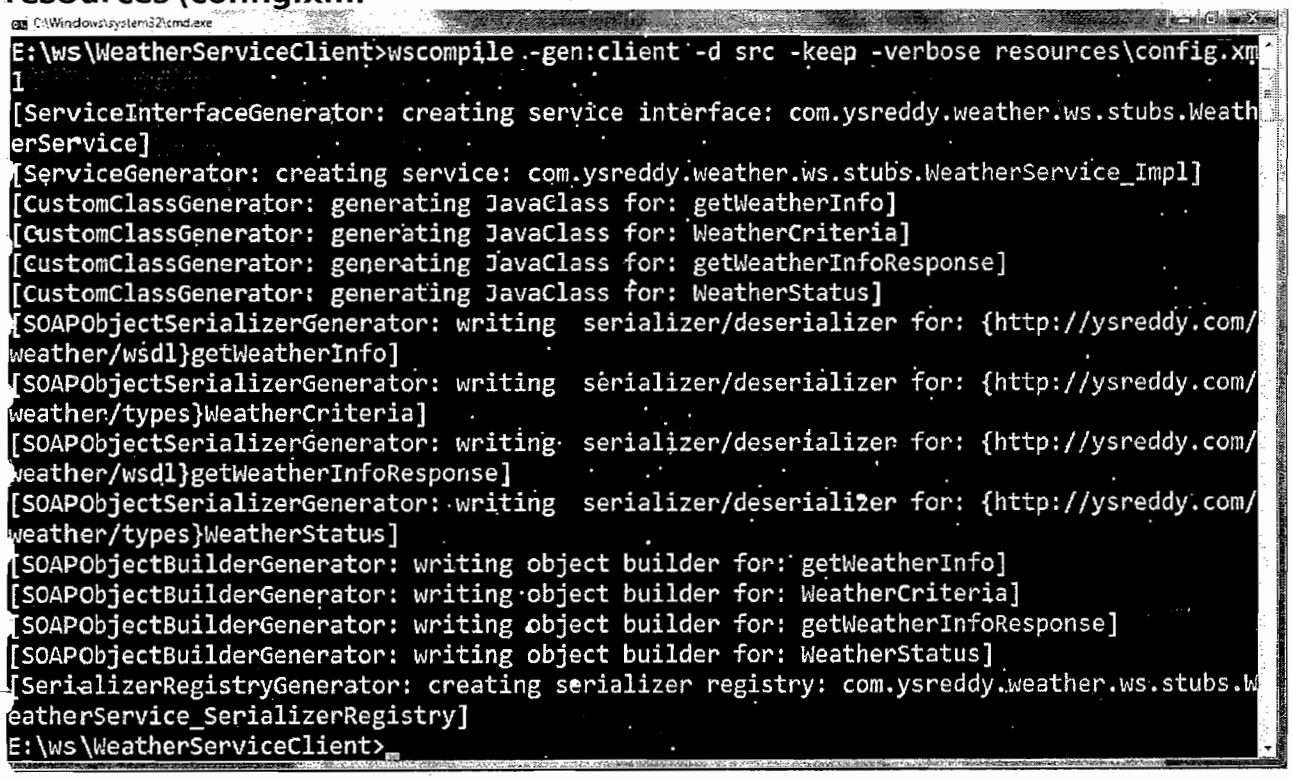
config.xml

```
1. <?xml version="1.0" encoding="UTF-8"?>
2.
3. <!--
4.  Copyright 2004 Sun Microsystems, Inc. All rights reserved.
5.  SUN PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
6. -->
7.
8. <configuration
9.     xmlns="http://java.sun.com/xml/ns/jax-rpc/ri/config">
10.       <wsdl
11.           location="http://localhost:8086/WeatherServiceWeb/ws/IWeatherInfo?WSDL"
12.           packageName="com.ysreddy.weather.ws.stubs">
13.       </wsdl>
14.   </configuration>
```

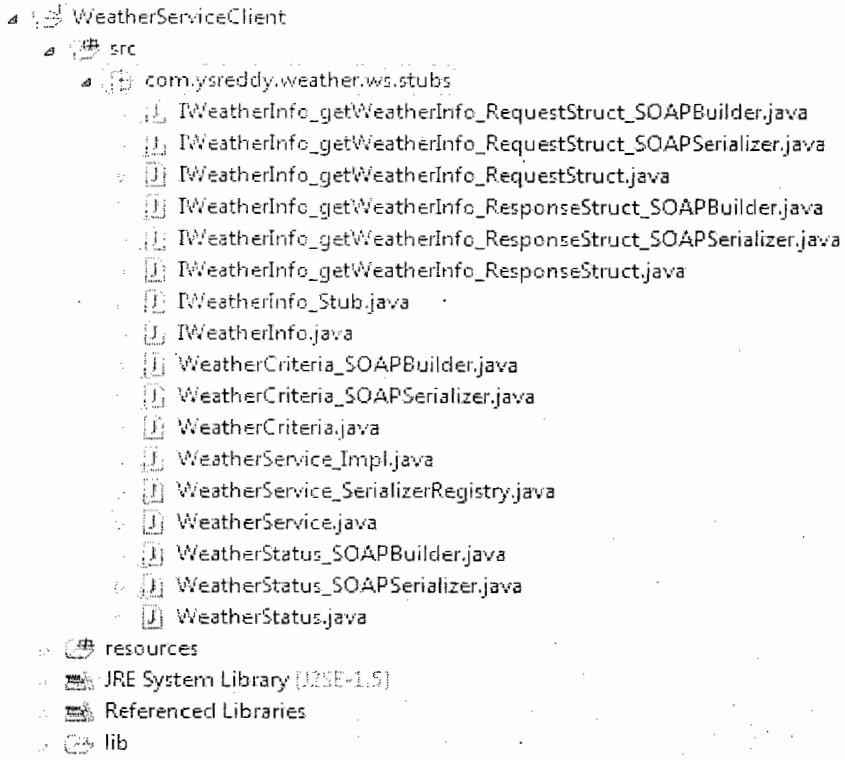
Step 3: Now open command prompt and navigate up to our project location as follows

Step 4: Use **wscompile** tool with **–gen:client** option to generate client side stubs

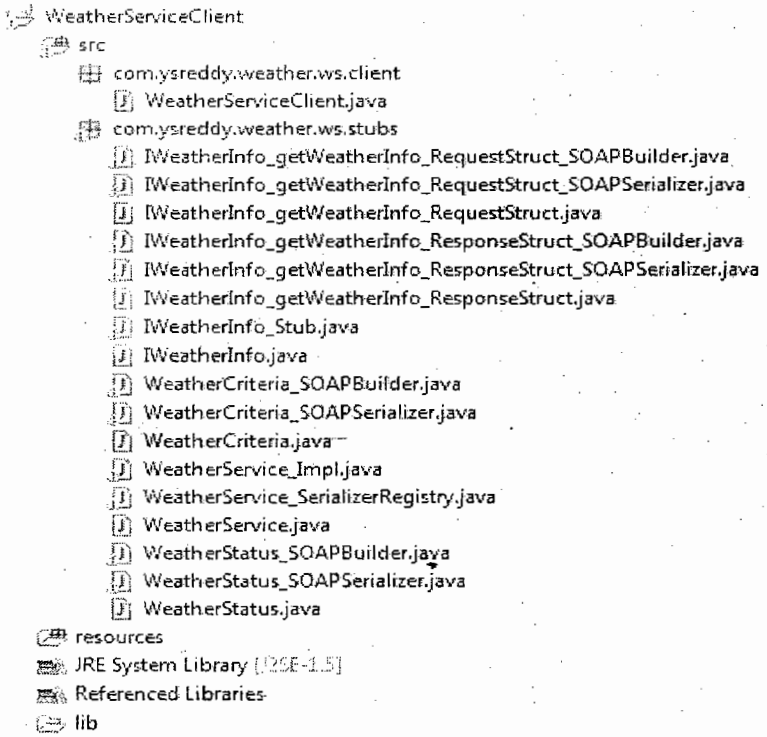
E:\ws\WeatherServiceClient>**wscompile -gen:client -d src -keep -verbose resources\config.xml**



Step 5: Now go and refresh our project, we can see generated stub classes in **com.ysreddy.weather.ws.stubs** package



Step 6: Now create one more **package com.ysreddy.weather.ws.client** and create class in that package with name "**WeatherServiceClient**" as follows.



WeatherServiceClient.java

```
1. package com.ysreddy.weather.ws.client;
2.
3. import java.rmi.RemoteException;
```

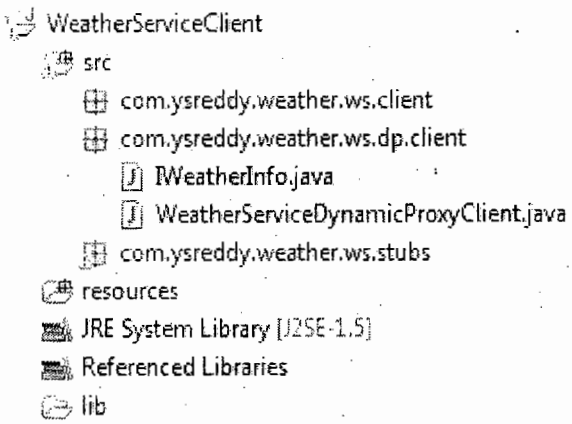
```
4.
5. import javax.xml.rpc.ServiceException;
6.
7. import com.ysreddy.weather.ws.stubs.IWeatherInfo;
8. import com.ysreddy.weather.ws.stubs.WeatherCriteria;
9. import com.ysreddy.weather.ws.stubs.WeatherService;
10.    import com.ysreddy.weather.ws.stubs.WeatherService_Impl;
11.    import com.ysreddy.weather.ws.stubs.WeatherStatus;
12.
13.    public class WeatherServiceClient {
14.          public static void main(String[] args)
15.                                  throws ServiceException, RemoteException {
16.                WeatherService weatherService = new WeatherService_Impl();
17.                IWeatherInfo weatherInfo =(IWeatherInfo)
18.                                       weatherService.getIWeatherInfoPort();
19.                WeatherCriteria weatherCriteria = new WeatherCriteria();
20.                weatherCriteria.setCity("Gunipalli");
21.                weatherCriteria.setState("AndhraPradesh");
22.                weatherCriteria.setCountry("India");
23.
24.                WeatherStatus weatherStatus=
25.                               weatherInfo.getWeatherInfo(weatherCriteria);
26.                System.out.println("Weather details...");
27.                System.out.println("Temperature:"+weatherStatus.getTemperature()
28.                System.out.println("Humidity:"+weatherStatus.getHumidity());
29.                System.out.println("Description:"+weatherStatus.getDescription()
30.          }
31.    }
```

Run the above program, we will get the following output

Weather details...
Temperature:101
Humidity:12.5
Description:Today may be rained or may not be ☺

2.Dynamic Proxy based client

In the above project create one more package with name "**com.ysreddy.weather.ws.dp.client**"



WeatherServiceDynamicProxyClient.java

```
1. package com.ysreddy.weather.ws.dp.client;
2.
3. import java.net.MalformedURLException;
4. import java.net.URL;
5. import java.rmi.RemoteException;
6.
7. import javax.xml.namespace.QName;
8. import javax.xml.rpc.Service;
9. import javax.xml.rpc.ServiceException;
10.   import javax.xml.rpc.ServiceFactory;
11.
12.   import com.ysreddy.weather.ws.stubs.WeatherCriteria;
13.   import com.ysreddy.weather.ws.stubs.WeatherStatus;
14.
15.   public class WeatherServiceDynamicProxyClient {
16.         final static String WSDL_URL =
17.                "http://localhost:8086/WeatherServiceWeb/ws/IWeatherInfo?WSDL";
18.         final static String SERVICE_NM = "WeatherService";
19.         final static String PORT_NM = "IWeatherInfoPort";
20.         final static String NAME_SPACE = "http://ysreddy.com/weather/wsdl";
21.
22.         public static void main(String[] args) throws ServiceException,
23.                     RemoteException, MalformedURLException {
24.
25.               ServiceFactory factory = ServiceFactory.newInstance();
26.               Service service = factory.createService(new URL(WSDL_URL),
27.                                                   new QName(NAME_SPACE, SERVICE_NM));
28.               IWeatherInfo weatherInfo = (IWeatherInfo)
29.                  service.getPort(new QName(NAME_SPACE, PORT_NM), IWeatherInfo.class);
30.
31.               WeatherCriteria weatherCriteria = new WeatherCriteria();
32.               weatherCriteria.setCity("Gunipalli");
33.               weatherCriteria.setState("AndhraPradesh");
34.               weatherCriteria.setCountry("India");
35.
36.             WeatherStatus weatherStatus= weatherInfo.getWeatherInfo(weatherCriteria);
37.               System.out.println("Weather details...");
38.               System.out.println("Temperature:"+weatherStatus.getTemperature());
39.               System.out.println("Humidity:"+weatherStatus.getHumidity());
40.               System.out.println("Description:"+weatherStatus.getDescription());
41.         }
42.   }
```

Run the above program, we will get the following output

Weather details...
Temperature:101
Humidity:12.5
Description:Today may be rained or may not be ☺

How to trace SOAP message in Eclipse IDE

In SOAP web service, each HTTP request or response encapsulates a SOAP envelope, these messages are easy to trace by using Eclipse IDE, build-in "**TCP/IP monitor**" tool. The idea is host another server in between the client and server to perform port forward function to intercept the HTTP traffic.

1. Normal SOAP envelope flows

In normal SOAP service, client send a HTTP request to server, and server send back a HTTP response to client directly.

```
1. Client ----> SOAP envelope ----> Server:8086
2. Server:8086 ----> SOAP envelope ---> Client
```

2. Intercepted SOAP envelope flows

To intercept SOAP envelope, you can host another server ("TcpMonitorServer") in between client and server, see new flows :

```
1. Client ----> SOAP envelope ----> TcpMonitorServer:8888
2. TcpMonitorServer:8888 --> SOAP envelope ---> Server:8086
3. Server:8086 ----> SOAP envelope ---> TcpMonitorServer:8888
4. TcpMonitorServer:8888 ----> SOAP envelope ---> Client
```

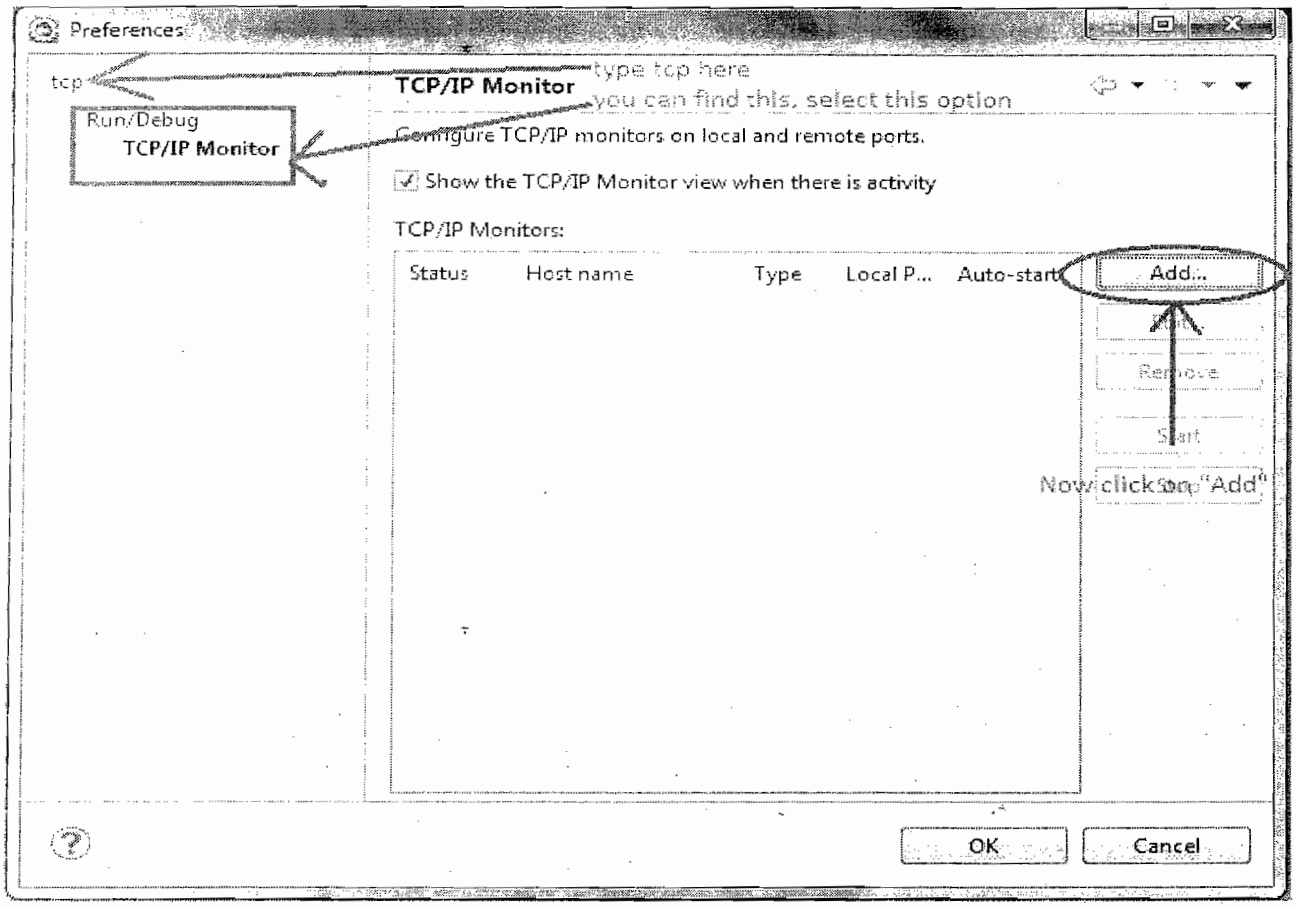
Note
This method required port changed in your web service client.

Eclipse IDE + TCP/IP Monitor

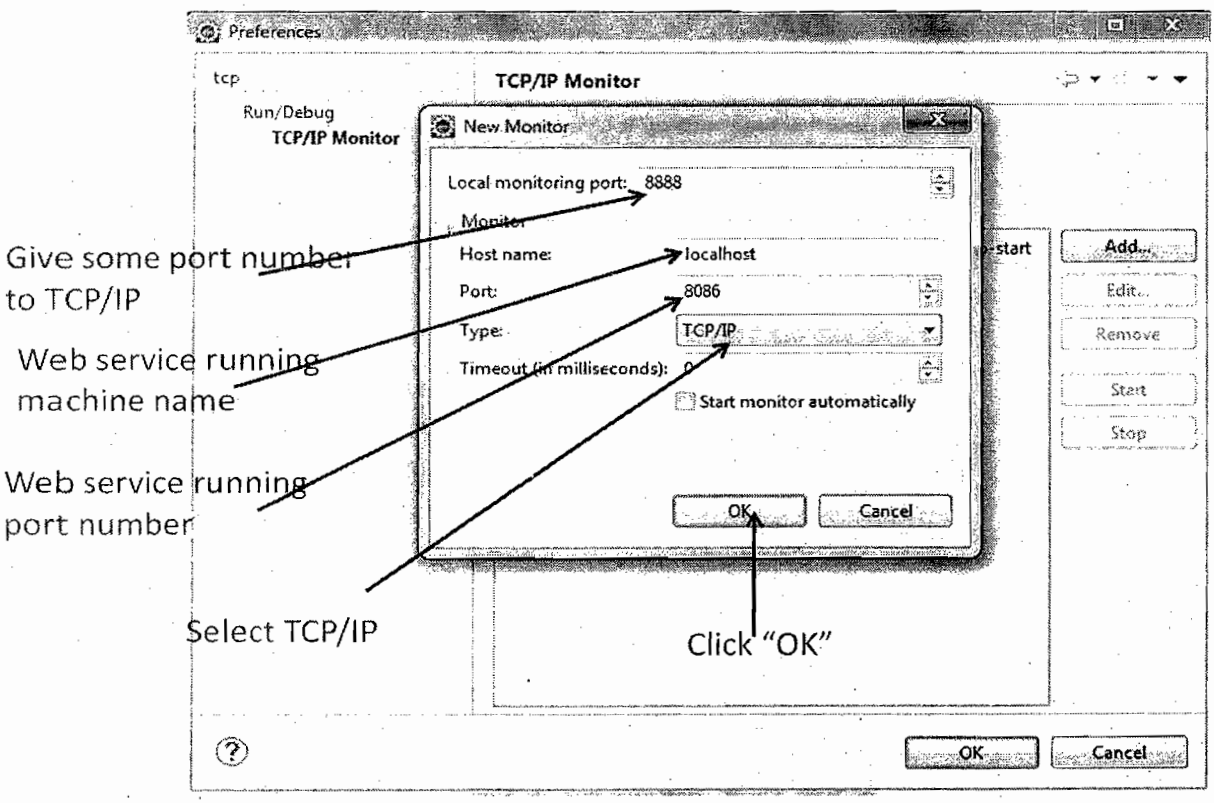
Eclipse IDE comes with a very easy to use traffic interceptor tool, known as "**TCP/IP Monitor**". In this tutorial, we show you how to enable this "TCP/IP Monitor" in Eclipse IDE, and also intercept the SOAP messages generated by web service.

Here's the steps to enable "TCP/IP Monitor" in Eclispe IDE.

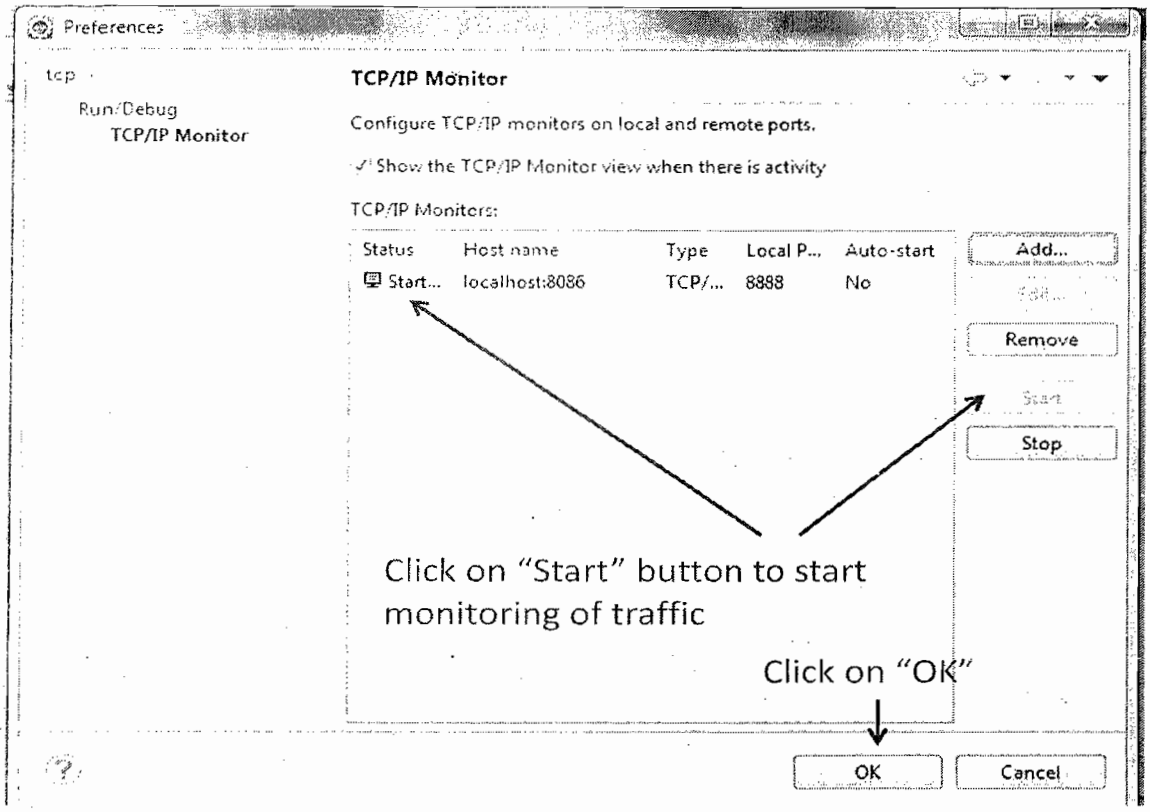
1. In IDE, access *Windows –> Preferences –> Run/Debug –> TCP/IP Monitor*



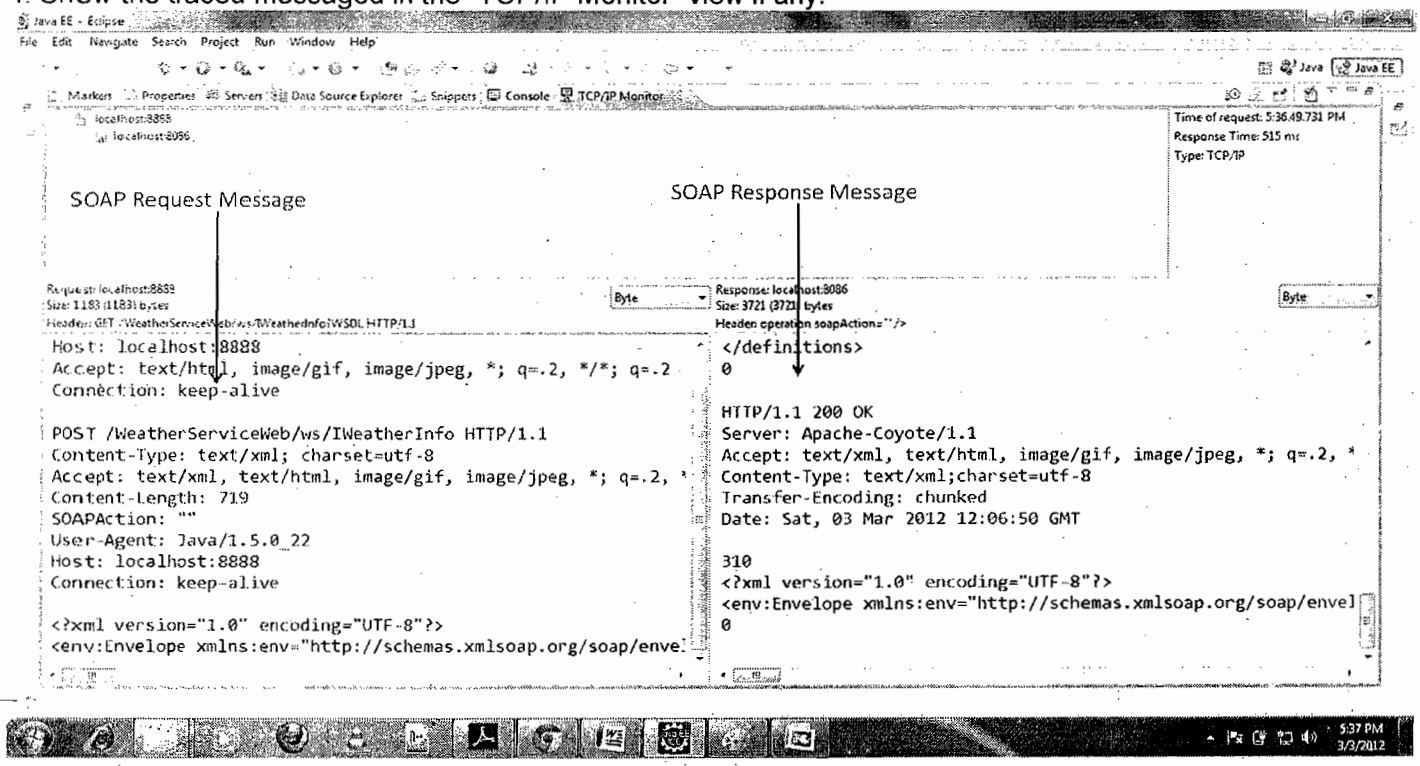
2. Fill in server information, and choose type = "**TCP/IP**"



3. Click on the "start" button to start tracing the web service traffic.



4. Show the traced messaged in the "TCP/IP Monitor" view if any.



Generated SOAP Request

POST /WeatherServiceWeb/ws/IWeatherInfo HTTP/1.1
Content-Type: text/xml; charset=utf-8

```
Accept: text/xml, text/html, image/gif, image/jpeg, *; q=.2, */*; q=.2
Content-Length: 719
SOAPAction: ""
User-Agent: Java/1.5.0_22
Host: localhost:8888
Connection: keep-alive

<?xml version="1.0" encoding="UTF-8"?>
<env:Envelope xmlns:env="http://schemas.xmlsoap.org/soap/envelope/"
        xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xmlns:enc="http://schemas.xmlsoap.org/soap/encoding/"
        env:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
        <env:Body>
                <ans1:getWeatherInfo xmlns:ans1="http://ysreddy.com/weather/wsdl">
                        <WeatherCriteria_1 href="#ID1" />
                </ans1:getWeatherInfo>
                <ans2:WeatherCriteria xmlns:ans2="http://ysreddy.com/weather/types"
                        id="ID1">
                        <city xsi:type="xsd:string">Gunipalli</city>
                        <country xsi:type="xsd:string">India</country>
                        <state xsi:type="xsd:string">AndhraPradesh</state>
                </ans2:WeatherCriteria>
        </env:Body>
</env:Envelope>
```

Generated SOAP Response

```
HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
Accept: text/xml, text/html, image/gif, image/jpeg, *; q=.2, */*; q=.2
Content-Type: text/xml;charset=utf-8
Transfer-Encoding: chunked
Date: Sat, 03 Mar 2012 07:40:44 GMT

<?xml version="1.0" encoding="UTF-8"?>
<env:Envelope xmlns:env="http://schemas.xmlsoap.org/soap/envelope/"
        xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xmlns:enc="http://schemas.xmlsoap.org/soap/encoding/" xmlns:ns0="http://ysreddy.com/weather/types"
        env:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
        <env:Body>
                <ans1:getWeatherInfoResponse xmlns:ans1="http://ysreddy.com/weather/wsdl">
                        <result href="#ID1" />
                </ans1:getWeatherInfoResponse>
                <ns0:WeatherStatus id="ID1" xsi:type="ns0:WeatherStatus">
                        <description xsi:type="xsd:string">Today may be rained or may not be</description>
                        <humidity xsi:type="xsd:double">12.5</humidity>
                        <temperature xsi:type="xsd:int">101</temperature>
                </ns0:WeatherStatus>
        </env:Body>
</env:Envelope>
```

Q.) Develop Webservice application using contract first approach, JAX-RPC API, JAX-RPC –SI implementation, Servlet as end point?

- Create a web project with any name, for example **CricketInfoContractFirstApp.** And add the JAX-RPC-SI implementation jars.
- As per this application we need to develop WSDL document first, but we are copying the previous WSDL document(**CricketScore.wsdl**) into WEB-INF folder of our application

CricketScore.wsdl

```
<?xml version="1.0" encoding="UTF-8"?>

<definitions name="CricketScore"
targetNamespace="http://sekharit.com/webservice/cricket/wsdl"
xmlns:tns="http://sekharit.com/webservice/cricket/wsdl"
xmlns="http://schemas.xmlsoap.org/wsdl/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:ns2="http://sekharit.com/webservice/cricket/types"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/">

        <types>
            <schema targetNamespace="http://sekharit.com/webservice/cricket/types"
                xmlns:tns="http://sekharit.com/webservice/cricket/types"
                xmlns:soap11-enc="http://schemas.xmlsoap.org/soap/encoding/"
                xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
                xmlns="http://www.w3.org/2001/XMLSchema">
                <import namespace="http://schemas.xmlsoap.org/soap/encoding/" />
                <complexType name="SearchCriteria">
                    <sequence>
                        <element name="name" type="string" />
                        <element name="playerId" type="int" />
                    </sequence>
                </complexType>
                <complexType name="PlayerRecord">
                    <sequence>
                        <element name="average" type="double" />
                        <element name="centuries" type="int" />
                        <element name="highestScore" type="int" />
                        <element name="name" type="string" />
                        <element name="totalScore" type="int" />
                    </sequence>
                </complexType>
            </schema>
        </types>

        <message name="ICricketInfo_getCricketPlayerInfo">
            <part name="SearchCriteria_1" type="ns2:SearchCriteria" />
        </message>
        <message name="ICricketInfo_getCricketPlayerInfoResponse">
            <part name="result" type="ns2:PlayerRecord" />
        </message>

        <portType name="ICricketInfo">
        <operation name="getCricketPlayerInfo" parameterOrder="SearchCriteria_1">
```

```
46.             <input message="tns:ICricketInfo_getCricketPlayerInfo" />
47.             <output message="tns:ICricketInfo_getCricketPlayerInfoResponse" />
48.         </operation>
49.         </portType>
50.
51.         <binding name="ICricketInfoBinding" type="tns:ICricketInfo">
52.             <soap:binding transport="http://schemas.xmlsoap.org/soap/http"
53.                 style="rpc" />
54.             <operation name="getCricketPlayerInfo">
55.                 <soap:operation soapAction="" />
56.                 <input>
57.                     <soap:body
58.                       encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
59.                       use="encoded"
60.                       namespace="http://sekharit.com/webservice/cricket/wsdl"/>
61.                 </input>
62.                 <output>
63.                     <soap:body
64.                      encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
65.                      use="encoded"
66.                      namespace="http://sekharit.com/webservice/cricket/wsdl" />
67.                 </output>
68.             </operation>
69.         </binding>
70.         <service name="CricketScore">
71.             <port name="ICricketInfoPort" binding="tns:ICricketInfoBinding">
72.                 <soap:address location="REPLACE_WITH_ACTUAL_URL" />
73.             </port>
74.         </service>
75.     </definitions>
```

➪ Now develop config.xml and configure WSDL documents in that file and apply that document to wscompile tool, then it will generate input, output objects, SEI, request serializers, response serializers...etc

Config.xml

```
1. <?xml version="1.0" encoding="UTF-8"?>
2.
3. <!--
4.  Copyright 2004 Sun Microsystems, Inc. All rights reserved.
5.  SUN PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
6. -->
7.
8. <configuration
9.     xmlns="http://java.sun.com/xml/ns/jax-rpc/ri/config">
10.       <wsdl
11.           location="./WebContent/WEB-INF/CricketScore.wsdl"
12.           packageName="com.sekharit.webservice">
13.       </wsdl>
14.   </configuration>
```

➪ Now go to your project and refresh it, should showing up the generated stubs/skeleton files

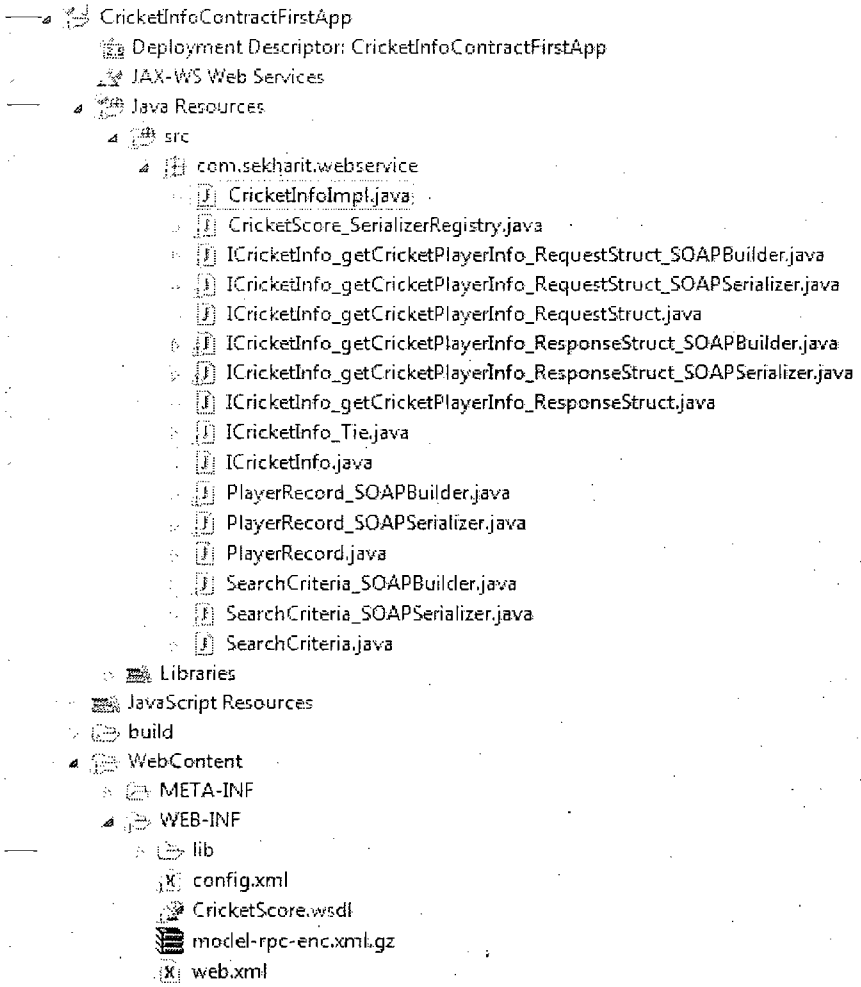
➪ But it will not generate implementation class of SEI interface. Because the implementation logic is not explained in WSDL document.

➪ So now we need to write our own implementation class for generated SEI interface.

CricketInfoImpl.java

```
1. package com.sekharit.webservices.service;
2.
3. public class CricketInfoImpl implements ICricketInfo {
4.     public PlayerRecord getCricketPlayerInfo(SearchCriteria criteria) {
5.           System.out.println("Cricter searching with ....");
6.           System.out.println("Id : " + criteria.getPlayerId());
7.           System.out.println("Name : " + criteria.getName());
8.
9.           PlayerRecord record = new PlayerRecord();
10.                record.setName("sachin");
11.                record.setCenturies(100);
12.                record.setAverage(47.5);
13.                record.setHighestScore(200);
14.                record.setTotalScore(17450);
15.
16.                return record;
17.          }
18.    }
```

➪ Now the structure of the project should be as follows



⇨ Now develop one more configuration file **jaxrpc-ri.xml** file where configure service details ... under **WEB-INF** directory.

Jaxrpc-ri.xml

```
<?xml version="1.0" encoding="UTF-8"?>

<!--
 Copyright 2004 Sun Microsystems, Inc. All rights reserved.
 SUN PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
-->

<webServices
    xmlns="http://java.sun.com/xml/ns/jax-rpc/ri/dd"
       version="1.0"
       targetNamespaceBase="http://sekharit.com/webservice/cricket/wsdl"
       typeNamespaceBase="http://sekharit.com/webservice/cricket/types"
       urlPatternBase="/ws">

       <endpoint
           name="CricketEndpoint"
           displayName="Cricket Web Service"
           description="Get Cricket Info Web Service"
           wsdl="/WEB-INF/CricketScore.wsdl"
           interface="com.sekharit.webservice.ICricketInfo"
           implementation="com.sekharit.webservice.CricketInfoImpl"
              model="/WEB-INF/model-rpc-enc.xml.gz"/>

       <endpointMapping
           endpointName="CricketEndpoint"
           urlPattern="/getPlayerRecord"/>

   </webServices>
```

⇨ Now export our project as **war** file onto external location by using option from **File → Export → WAR file.**

⇨ Now navigate to the directory where we exported our project as war file and run **wsdeploy** command.

E:\SekharReddy\WEB-SERVICES\WS\CricketInfoContractFirstApp>**wscompile -gen:server -cp build/classes -d src -keep -model model-rpc-enc.xml.gz -verbose WebContent\WEB-INF\config.xml**

⇨ Now unzip the **targer.war**, you should be able to find **jaxrpc-ri-runtime.xml** and **web.xml.**

⇨ Copy these both files into **WEB-INF** directory. Deploy the project and start sever.

You can find the following changes in web.xml

web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://java.sun.com/xml/ns/javaee"
    xmlns:web="http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" id="WebApp_ID"
    version="2.5"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd">
    <display-name>CricketInfoContractFirstApp</display-name>
```

```
    <listener>
    <listener-class>com.sun.xml.rpc.server.http.JAXRPCContextListener</listener-class>
        </listener>
        <servlet>
            <servlet-name>CricketEndpoint</servlet-name>
        <servlet-class>com.sun.xml.rpc.server.http.JAXRPCServlet</servlet-class>
            <load-on-startup>1</load-on-startup>
        </servlet>
        <servlet-mapping>
            <servlet-name>CricketEndpoint</servlet-name>
            <url-pattern>/getPlayerRecord</url-pattern>
        </servlet-mapping>
        <welcome-file-list>
            <welcome-file>index.html</welcome-file>
            <welcome-file>index.htm</welcome-file>
            <welcome-file>index.jsp</welcome-file>
            <welcome-file>default.html</welcome-file>
            <welcome-file>default.htm</welcome-file>
            <welcome-file>default.jsp</welcome-file>
        </welcome-file-list>
    </web-app>
```

NOTE: Open jaxrpc-ri-runtime.xml file and make sure that **tie="com.sekharit.webservice.ICricketInfo_Tie"** is available.

⇨ Now deploy your project into servlet container like tomcat and start server.

⇨ Now access your webservice wsdl with the below url format in a web browser.
http://<domain>:<port>/<context-root>/<servletpath/url pattern of your service>?wsdl

http://localhost:8086/CricketInfoContractFirstApp/getPlayerRecord?WSDL

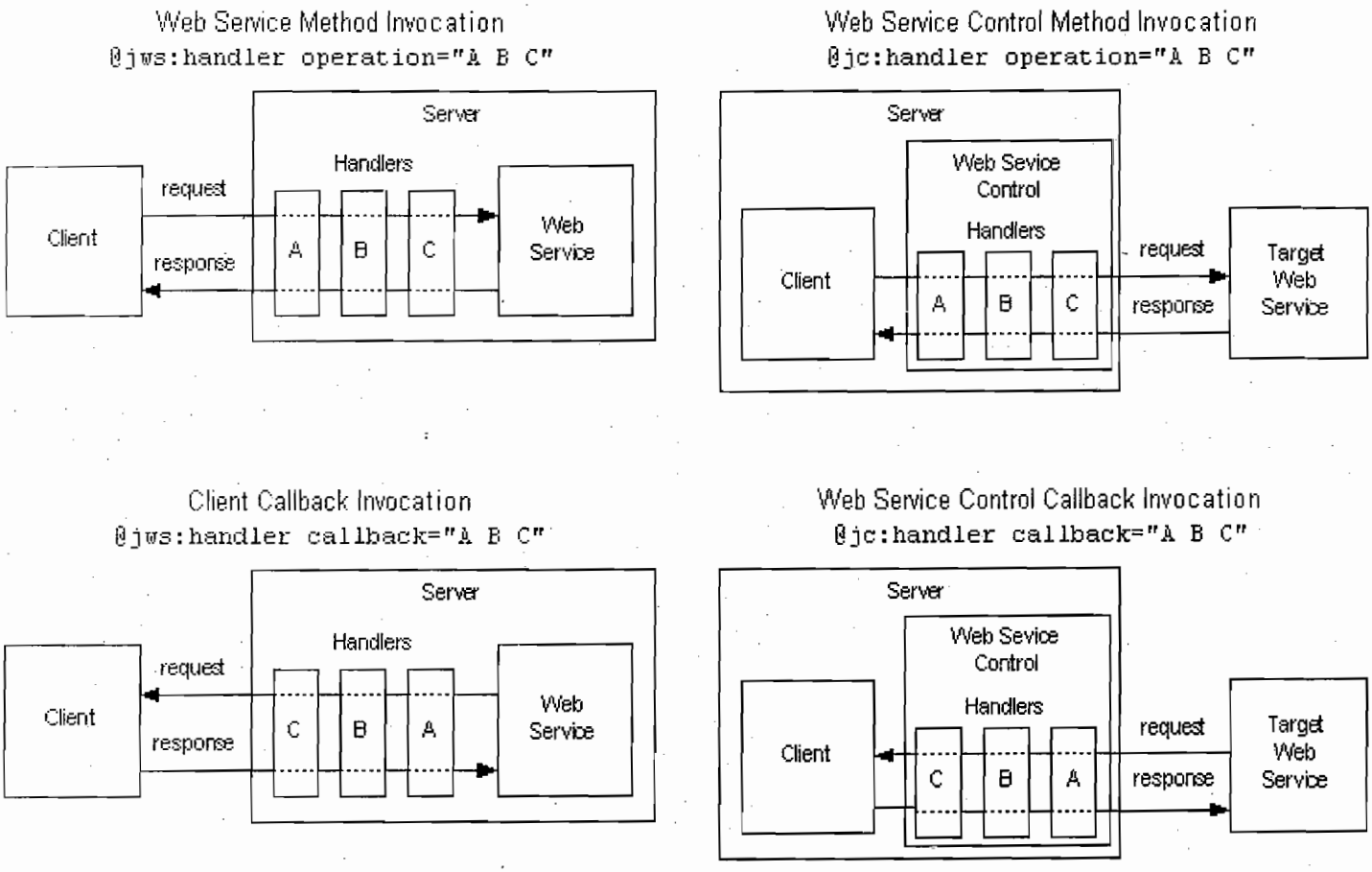
```
<?xml version="1.0" encoding="UTF-8" ?>
- <definitions xmlns="http://schemas.xmlsoap.org/wsdl/" xmlns:tns="http://sekharit.com/webservice/cricket/wsdl" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:ns2="http://sekharit.com/webservice/cricket/types" xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/" name="CricketScore"
  targetNamespace="http://sekharit.com/webservice/cricket/wsdl">
- <types>
  - <schema xmlns="http://www.w3.org/2001/XMLSchema" xmlns:tns="http://sekharit.com/webservice/cricket/types" xmlns:soap11-
      enc="http://schemas.xmlsoap.org/soap/encoding/" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
      targetNamespace="http://sekharit.com/webservice/cricket/types">
      <import namespace="http://schemas.xmlsoap.org/soap/encoding/" />
    - <complexType name="SearchCriteria">
      - <sequence>
```

Specifying SOAP Handlers for a Web Service

When designing a web service that accepts or returns SOAP messages, you may specify that the SOAP messages should be processed by *message handlers*. Handlers on incoming messages are invoked before the message is delivered to the web service operation, and handlers on outgoing messages are invoked after the web service operation has completed. The web service itself is unaware of the presence of handlers. SOAP message handlers are sometimes referred to as *interceptors*.

SOAP message handlers can be used for a variety of purposes. Examples include message logging, audit trails, and message transformations. Message handlers should*not* be used to implement security or encryption or to manage the redirection of messages; those facilities are provided by other mechanisms.

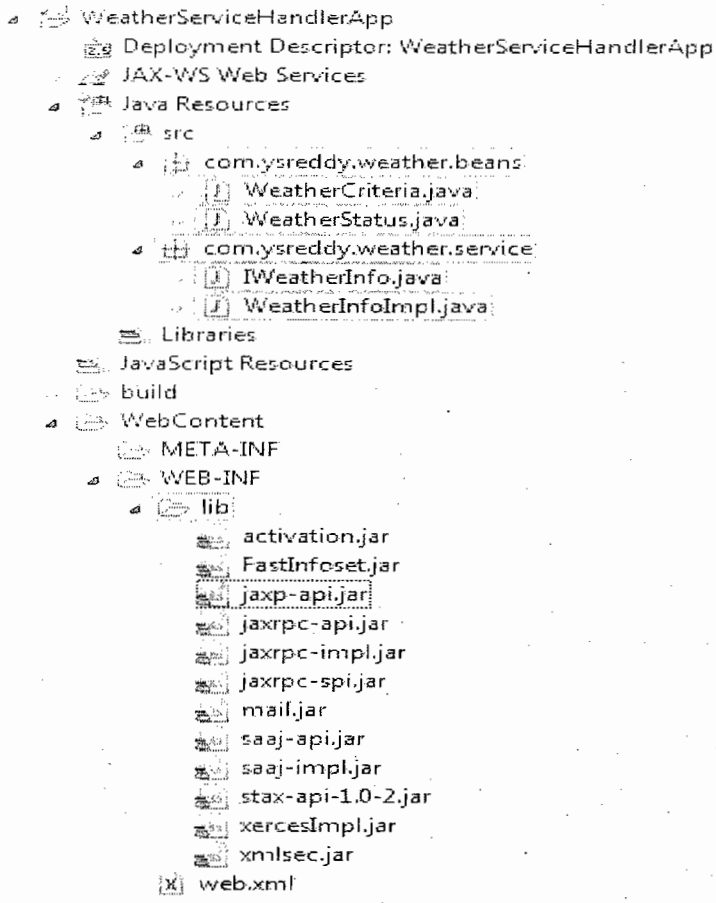
JAX-RPC and SOAP Handlers



Q.) Develop Webservice application where you can apply Handler to webservice at server side, which can print input and output SOAP message?

- ⇨ Create a new **Dynamic Web Project** for example with name **WeatherServiceHandlerApp**. While creating your project make sure you configure the **target runtime**(configure tomcat in eclipse).
- ⇨ After setting up the project copy all the required jar bundles in to project **WebContent/WEBINF/lib** directory.

⇨ Develop Input class, output class, SEI interface and SEI interface implementaion class.



WeatherCriteria.java

```
1. package com.ysreddy.weather.beans;
2.
3. import java.io.Serializable;
4.
5. public class WeatherCriteria implements Serializable {
6.     private String city;
7.     private String state;
8.     private String country;
9.
10.          // setters & getters
11.
12.    }
```

WeatherStatus.java

```
1. package com.ysreddy.weather.beans;
2.
3. import java.io.Serializable;
4.
5. public class WeatherStatus implements Serializable {
6.     private int temperature;
7.     private double humidity;
8.     private String description;
9.
```

```
10.         // setters & getters
11.
12.    }
```

IWeatherInfo.java

```
1. package com.ysreddy.weather.service;
2.
3. import java.rmi.Remote;
4. import java.rmi.RemoteException;
5.
6. import com.ysreddy.weather.beans.WeatherCriteria;
7. import com.ysreddy.weather.beans.WeatherStatus;
8.
9. public interface IWeatherInfo extends Remote {
10.         public WeatherStatus getWeatherInfo(WeatherCriteria weatherCriteria)
11.                                                                 throws RemoteException;
12.    }
```

WeatherInfoImpl.java

```
1. package com.ysreddy.weather.service;
2.
3. import com.ysreddy.weather.beans.WeatherCriteria;
4. import com.ysreddy.weather.beans.WeatherStatus;
5.
6. public class WeatherInfoImpl implements IWeatherInfo {
7.     public WeatherStatus getWeatherInfo(WeatherCriteria weatherCriteria){
8.
9.           if (weatherCriteria != null) {
10.                    System.out.println("city :" + weatherCriteria.getCity());
11.                    System.out.println("state :" + weatherCriteria.getState());
12.                    System.out.println("country :" + weatherCriteria.getCountry());
13.                }
14.                WeatherStatus status = new WeatherStatus();
15.                status.setTemperature(101);
16.                status.setHumidity(12.5);
17.                status.setDescription("Today may be rained or may not be");
18.                return status;
19.          }
20.    }
```

➪ Develop **config.xml** and configure our SEI, SEI implementation, types namespace and target namespace values. And then run **wscompile** by inputting this **config.xml** file

config.xml

```
1. <?xml version="1.0" encoding="UTF-8"?>
2.     <!--
3.           Copyright 2004 Sun Microsystems, Inc. All rights reserved. SUN
4.           PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
5.     -->
6. <configuration xmlns="http://java.sun.com/xml/ns/jax-rpc/ri/config">
7.     <service name="WeatherService"
8.           targetNamespace="http://ysreddy.com/weather/wsdl"
9.           typeNamespace="http://ysreddy.com/weather/types"
10.                packageName="com.ysreddy.weather.service.binding">
```

```
11.             <interface name="com.ysreddy.weather.service.IWeatherInfo"
12.                     servantName="com.ysreddy.weather.service.WeatherInfoImpl" />
13.         </service>
14.   </configuration>
```

```
C:\Windows\system32\cmd.exe
E:\SekharReddy\WEB-SERVICES\WS\WeatherServiceHandlerApp>wscompile -gen:server -c
p build/classes -d src -keep -model model-rpc-enc.xml.gz -verbose WebContent\WEB
-INF\config.xml
[creating model: WeatherService]
[creating service: WeatherService]
[creating port: com.ysreddy.weather.service.IWeatherInfo]
[creating operation: getWeatherInfo]
[CustomClassGenerator: generating JavaClass for: getWeatherInfo]
[CustomClassGenerator: generating JavaClass for: getWeatherInfoResponse]
[SOAPObjectSerializerGenerator: writing serializer/deserializer for: {http://ys
reddy.com/weather/types}getWeatherInfo]
[SOAPObjectSerializerGenerator: writing serializer/deserializer for: {http://ys
reddy.com/weather/types}WeatherCriteria]
[SOAPObjectSerializerGenerator: writing serializer/deserializer for: {http://ys
reddy.com/weather/types}getWeatherInfoResponse]
[SOAPObjectSerializerGenerator: writing serializer/deserializer for: {http://ys
reddy.com/weather/types}WeatherStatus]
[SOAPObjectBuilderGenerator: writing object builder for: getWeatherInfo]
[SOAPObjectBuilderGenerator: writing object builder for: WeatherCriteria]
[SOAPObjectBuilderGenerator: writing object builder for: getWeatherInfoResponse]
[SOAPObjectBuilderGenerator: writing object builder for: WeatherStatus]
[SerializerRegistryGenerator: creating serializer registry: com.ysreddy.weather.
```

- ⇨ Now go to your project and refresh it, should start showing up the generated stubs/skeleton files
- ⇨ Now drag and drop WSDL file and model file into **WEB-INF** directory.
- ⇨ Now deveop Handler class which prints request SOAP message and response SOAP message on to the console.

LogSOAPMessageHandler.java

```
1. package com.ysreddy.common.handler;
2.
3. import java.io.IOException;
4.
5. import javax.xml.namespace.QName;
6. import javax.xml.rpc.handler.Handler;
7. import javax.xml.rpc.handler.HandlerInfo;
8. import javax.xml.rpc.handler.MessageContext;
9. import javax.xml.soap.SOAPException;
10.   import javax.xml.soap.SOAPMessage;
11.
12.   import com.sun.xml.rpc.soap.message.SOAPMessageContext;
13.
14.   public class LogSOAPMessageHandler implements Handler {
15.
16.         public void destroy() {
17.               System.out.println("...AuthHandler.destroy()...");
18.         }
19.
20.         public QName[] getHeaders() {
21.               System.out.println("...AuthHandler.getHeaders()...");
22.               return null;
23.         }
24.
25.         public boolean handleFault(MessageContext arg0) {
```

```
26.               System.out.println("...AuthHandler.handleFault()...");
27.               return false;
28.         }
29.
30.         public boolean handleRequest(MessageContext ctx) {
31.               System.out.println("...AuthHandler.handleRequest()...");
32.               SOAPMessageContext sctx = (SOAPMessageContext) ctx;
33.               SOAPMessage message = sctx.getMessage();
34.               try {
35.                     message.writeTo(System.out);
36.               } catch (SOAPException e) {
37.                     e.printStackTrace();
38.               } catch (IOException e) {
39.                     e.printStackTrace();
40.               }
41.               return true;
42.         }
43.
44.         public boolean handleResponse(MessageContext ctx) {
45.               System.out.println("...AuthHandler.handleResponse()...");
46.               SOAPMessageContext sctx = (SOAPMessageContext) ctx;
47.               SOAPMessage message = sctx.getMessage();
48.               try {
49.                     message.writeTo(System.out);
50.               } catch (SOAPException e) {
51.                     e.printStackTrace();
52.               } catch (IOException e) {
53.                     e.printStackTrace();
54.               }
55.               return true;
56.         }
57.
58.         public void init(HandlerInfo arg0) {
59.               System.out.println("...AuthHandler.init()...");
60.         }
61.
62. }
```

⇨ Now develop one more configuration file **jaxrpc-ri.xml** file where configure service details and handler details... under **WEB-INF** directory.

jaxrpc-ri.xml

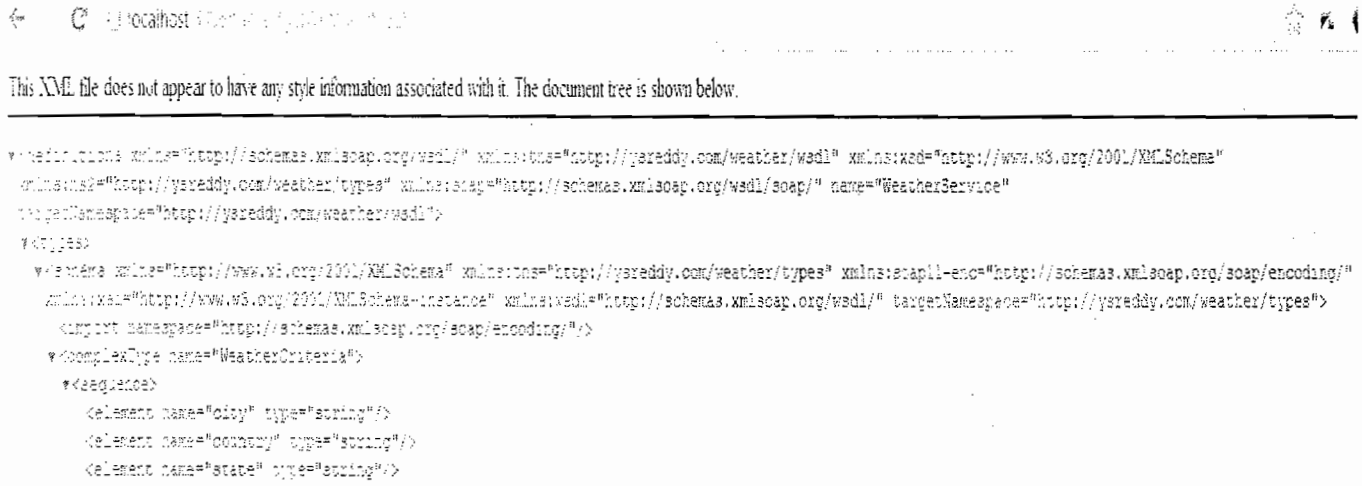
```
1. <?xml version="1.0" encoding="UTF-8"?>
2.
3. <!--
4.  Copyright 2004 Sun Microsystems, Inc. All rights reserved.
5.  SUN PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
6. -->
7.
8. <webServices
9.     xmlns="http://java.sun.com/xml/ns/jax-rpc/ri/dd"
10.       version="1.0"
11.       targetNamespaceBase="http://ysreddy.com/weather/wsdl"
12.       typeNamespaceBase="http://ysreddy.com/weather/types"
```

```
13.       urlPatternBase="/ws">
14.
15.       <endpoint
16.           name="Weather"
17.           displayName="Weather Web Service"
18.           description="Get Weather Web Service"
19.           wsdl="/WEB-INF/WeatherService.wsdl"
20.           interface="com.ysreddy.weather.service.IWeatherInfo"
21.           implementation="com.ysreddy.weather.service.WeatherInfoImpl"
22.              model="/WEB-INF/model-rpc-enc.xml.gz">
23.              <handlerChains>
24.               <chain runAt="server">
25.               <handler className="com.ysreddy.common.handler.LogSOAPMessageHandler"/>
26.               </chain>
27.           </handlerChains>
28.         </endpoint>
29.
30.       <endpointMapping
31.           endpointName="Weather"
32.           urlPattern="/getWeather"/>
33.
34.   </webServices>
```

- ⇨ Now export our project as **war** file onto external location by using option from **File → Export → WAR file**.
- ⇨ Now rename the exported war file as ZIP file... Now go to classes folder and delete Tile class... Then again rename that file as war file...
- ⇨ Now navigate to the directory where we exported our project as war file and run **wsdeploy** command.

```
C:\Windows\system32\cmd.exe
E:\temp>wsdeploy -verbose -o target.war WeatherServiceHandlerApp.war
info: created temporary directory: C:\Users\nit\AppData\Local\Temp\jaxrpc-deploy
-4f1c0
[CustomClassGenerator: Class com.ysreddy.weather.service.IWeatherInfo_getWeather
Info_RequestStruct exists. Not overriding.]
[CustomClassGenerator: Class com.ysreddy.weather.service.IWeatherInfo_getWeather
Info_ResponseStruct exists. Not overriding.]
[SOAPObjectSerializerGenerator: writing  serializer/deserializer for: {http://ys
reddy.com/weather/types}getWeatherInfo]
[SOAPObjectSerializerGenerator: writing  serializer/deserializer for: {http://ys
reddy.com/weather/types}WeatherCriteria]
[SOAPObjectSerializerGenerator: writing  serializer/deserializer for: {http://ys
reddy.com/weather/types}getWeatherInfoResponse]
[SOAPObjectSerializerGenerator: writing  serializer/deserializer for: {http://ys
reddy.com/weather/types}WeatherStatus]
[SOAPObjectBuilderGenerator: writing object builder for: getWeatherInfo]
[SOAPObjectBuilderGenerator: writing object builder for: WeatherCriteria]
[SOAPObjectBuilderGenerator: writing object builder for: getWeatherInfoResponse]

[SOAPObjectBuilderGenerator: writing object builder for: WeatherStatus]
[SerializerRegistryGenerator: Class com.ysreddy.weather.service.binding.WeatherS
ervice_SerializerRegistry exists. Not overriding.]
Note: C:\Users\nit\AppData\Local\Temp\jaxrpc-deploy-4f1c0\WEB-INF\classes\com\ys
reddy\weather\service\IWeatherInfo_Tie.java uses unchecked or unsafe operations.
```

- ⇨ Now deploy the generated **target.war** file into tomcat container... Now try to get WSDL document with browser...



➪ Now try to send the request to the service, then we can find following request SOAP message and response SOAP message on the console of the tomcat server...

Request SOAP message

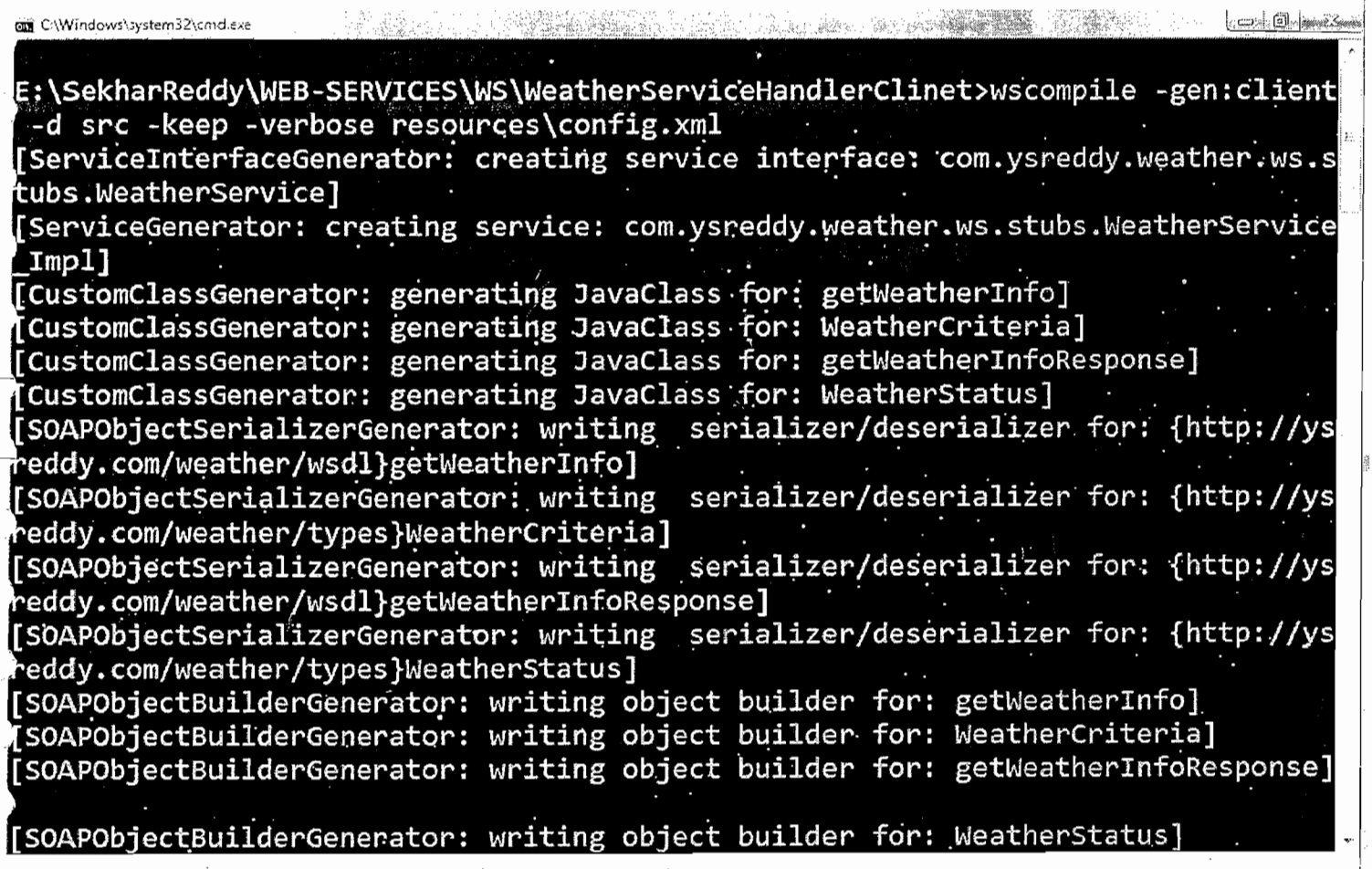
```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
        xmlns:wsdl="http://ysreddy.com/weather/wsdl" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
        <soapenv:Header />
        <soapenv:Body>
                <wsdl:getWeatherInfo soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
                        <WeatherCriteria_1 xmlns:typ="http://ysreddy.com/weather/types"
                                xsi:type="typ:WeatherCriteria">
                                <city xsi:type="xsd:string">Gunipalli</city>
                                <country xsi:type="xsd:string">India</country>
                                <state xsi:type="xsd:string">AP</state>
                        </WeatherCriteria_1>
                </wsdl:getWeatherInfo>
        </soapenv:Body>
</soapenv:Envelope>
```

Response SOAP message

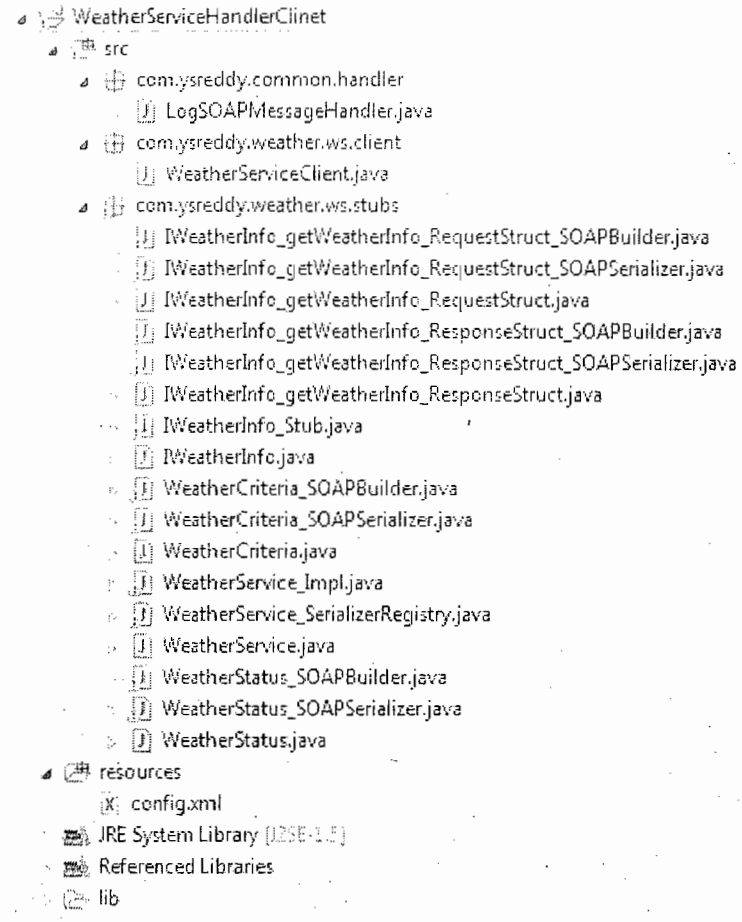
```
<env:Envelope xmlns:env="http://schemas.xmlsoap.org/soap/envelope/"
        xmlns:enc="http://schemas.xmlsoap.org/soap/encoding/" xmlns:ns0="http://ysreddy.com/weather/types"
        xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        env:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
        <env:Body>
                <ans1:getWeatherInfoResponse xmlns:ans1="http://ysreddy.com/weather/wsdl">
                        <result href="#ID1" />
                </ans1:getWeatherInfoResponse>
                <n s0:WeatherStatus id="ID1" xsi:type="ns0:WeatherStatus">
                        <description xsi:type="xsd:string">Today may be rained or may not be</description>
                        <humidity xsi:type="xsd:double">12.5</humidity>
                        <temperature xsi:type="xsd:int">101</temperature>
                </ns0:WeatherStatus>
        </env:Body>
</env:Envelope>
```

2.) Develop Webservice client application where you can apply Handler to webservice at client side, which can print input and output SOAP message?

- ⇨ Create a java project, for example with name "**WeatherServiceHandlerClinet**" and add the required jar files.
- ⇨ Now create a source folder with name "**resourses**" in the project, which was created in the previous step.
- ⇨ Now develop Handler class where we can print input and ouput SOAP messages.
- ⇨ Now create a file called "config.xml", in that file configure, WSDL url, package name where stubs has to create, and configre Handler class.
- ⇨ Now run the wscompile command to create the client side artifacts...



- ⇨ Now develop stub based client, now the finally project lools as follows...



LogSOAPMessageHandler.java

```
1. package com.ysreddy.common.handler;
2.
3. import java.io.IOException;
4.
5. import javax.xml.namespace.QName;
6. import javax.xml.rpc.handler.Handler;
7. import javax.xml.rpc.handler.HandlerInfo;
8. import javax.xml.rpc.handler.MessageContext;
9. import javax.xml.soap.SOAPException;
10.   import javax.xml.soap.SOAPMessage;
11.
12.   import com.sun.xml.rpc.soap.message.SOAPMessageContext;
13.
14.   public class LogSOAPMessageHandler implements Handler {
15.
16.         public void destroy() {
17.               System.out.println("...AuthHandler.destroy()...");
18.         }
19.
20.         public QName[] getHeaders() {
21.               System.out.println("...AuthHandler.getHeaders()...");
22.               return null;
23.         }
24.
25.         public boolean handleFault(MessageContext arg0) {
```

```
26.               System.out.println("...AuthHandler.handleFault()...");
27.               return false;
28.         }
29.
30.         public boolean handleRequest(MessageContext ctx) {
31.               System.out.println("...AuthHandler.handleRequest()...");
32.               SOAPMessageContext sctx = (SOAPMessageContext) ctx;
33.               SOAPMessage message = sctx.getMessage();
34.               try {
35.                     message.writeTo(System.out);
36.               } catch (SOAPException e) {
37.                     e.printStackTrace();
38.               } catch (IOException e) {
39.                     e.printStackTrace();
40.               }
41.               return true;
42.         }
43.
44.         public boolean handleResponse(MessageContext ctx) {
45.               System.out.println("...AuthHandler.handleResponse()...");
46.               SOAPMessageContext sctx = (SOAPMessageContext) ctx;
47.               SOAPMessage message = sctx.getMessage();
48.               try {
49.                     message.writeTo(System.out);
50.               } catch (SOAPException e) {
51.                     e.printStackTrace();
52.               } catch (IOException e) {
53.                     e.printStackTrace();
54.               }
55.               return true;
56.         }
57.
58.         public void init(HandlerInfo arg0) {
59.               System.out.println("...AuthHandler.init()...");
60.         }
61.
62.   }
```

WeatherServiceClient.java

```
1. package com.ysreddy.weather.ws.client;
2. import java.rmi.RemoteException;
3.
4. import javax.xml.rpc.ServiceException;
5.
6. import com.ysreddy.weather.ws.stubs.IWeatherInfo;
7. import com.ysreddy.weather.ws.stubs.WeatherCriteria;
8. import com.ysreddy.weather.ws.stubs.WeatherService;
9. import com.ysreddy.weather.ws.stubs.WeatherService_Impl;
10.   import com.ysreddy.weather.ws.stubs.WeatherStatus;
11.
12.   public class WeatherServiceClient {
13.   public static void main(String[] args) throws ServiceException, RemoteException {
14.               WeatherService weatherService = new WeatherService_Impl();
15.   IWeatherInfo weatherInfo =(IWeatherInfo) weatherService.getIWeatherInfoPort();
16.
```

```
17.
18.
19.             WeatherCriteria weatherCriteria = new WeatherCriteria();
20.             weatherCriteria.setCity("Gunipalli");
21.             weatherCriteria.setState("AndhraPradesh");
22.             weatherCriteria.setCountry("India");
23.
24.         WeatherStatus weatherStatus=  weatherInfo.getWeatherInfo(weatherCriteria);
25.             System.out.println("Weather details...");
26.             System.out.println("Temperature:"+weatherStatus.getTemperature());
27.             System.out.println("Humidity:"+weatherStatus.getHumidity());
28.             System.out.println("Description:"+weatherStatus.getDescription());
29.         }
30.   }
```

⇨ Now Run the client application, then we can find the following following request and response SOAP messages.

Request SOAP message

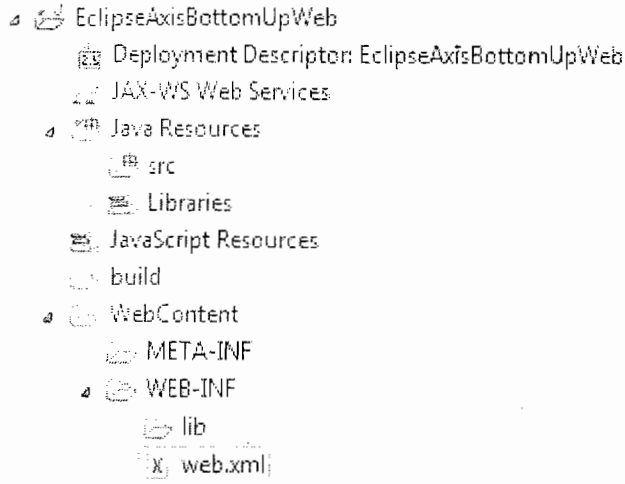
```
<env:Envelope xmlns:env="http://schemas.xmlsoap.org/soap/envelope/"
        xmlns:enc="http://schemas.xmlsoap.org/soap/encoding/" xmlns:ns0="http://ysreddy.com/weather/wsdl"
        xmlns:ns1="http://ysreddy.com/weather/types" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        env:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
        <env:Body>
                <ns0:getWeatherInfo>
                        <WeatherCriteria_1 href="#ID1" />
                </ns0:getWeatherInfo>
                <ns1:WeatherCriteria id="ID1" xsi:type="ns1:WeatherCriteria">
                        <city xsi:type="xsd:string">Gunipalli</city>
                        <country xsi:type="xsd:string">India</country>
                        <state xsi:type="xsd:string">AndhraPradesh</state>
                </ns1:WeatherCriteria>
        </env:Body>
</env:Envelope>
```

Response SOAP message

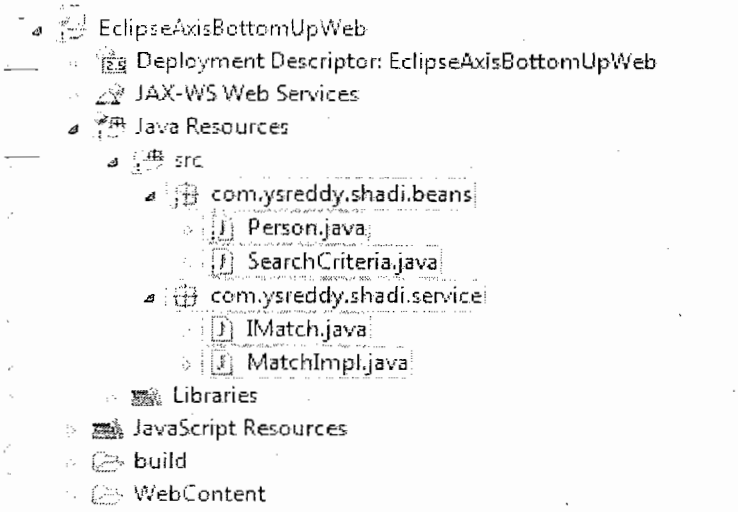
```
<env:Envelope xmlns:env="http://schemas.xmlsoap.org/soap/envelope/"
        xmlns:enc="http://schemas.xmlsoap.org/soap/encoding/" xmlns:ns0="http://ysreddy.com/weather/types"
        xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        env:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
        <env:Body>
                <ans1:getWeatherInfoResponse xmlns:ans1="http://ysreddy.com/weather/wsdl">
                        <result href="#ID1" />
                </ans1:getWeatherInfoResponse>
                <ns0:WeatherStatus id="ID1" xsi:type="ns0:WeatherStatus">
                        <description xsi:type="xsd:string">Today may be rained or may not be
                        </description>
                        <humidity xsi:type="xsd:double">12.5</humidity>
                        <temperature xsi:type="xsd:int">101</temperature>
                </ns0:WeatherStatus>
        </env:Body>
</env:Envelope>
```

Q.) Develop Web service using bottom-up approach, with eclipse IDE(JAX-RPC API, AXIS IMPLEMENTATION)?

Step 1: Create a **"Dynamic Web Project"** with some name like **"EclipseAxisBottomUpWeb"**



Step 2: create input and output objects, SEI interface, Implementation class



SearchCriteria.java

```
1. package com.ysreddy.shadi.beans;
2.
3. import java.io.Serializable;
4.
5. public class SearchCriteria implements Serializable {
6.     private int age;
7.     private double salary;
8.     private String color;
9.
10.         public int getAge() {
11.               return age;
12.         }
13.
14.         public void setAge(int age) {
15.               this.age = age;
```

```
16.         }
17.
18.         public double getSalary() {
19.               return salary;
20.         }
21.
22.         public void setSalary(double salary) {
23.               this.salary = salary;
24.         }
25.
26.         public String getColor() {
27.               return color;
28.         }
29.
30.         public void setColor(String color) {
31.               this.color = color;
32.         }
33.
34.   }
```

Person.java

```
1. package com.ysreddy.shadi.beans;
2.
3. import java.io.Serializable;
4.
5. public class Person implements Serializable {
6.     private String name;
7.     private String city;
8.
9.     public String getName() {
10.              return name;
11.        }
12.
13.        public void setName(String name) {
14.              this.name = name;
15.        }
16.
17.        public String getCity() {
18.              return city;
19.        }
20.
21.        public void setCity(String city) {
22.              this.city = city;
23.        }
24.
25.   }
```

IMatch.java

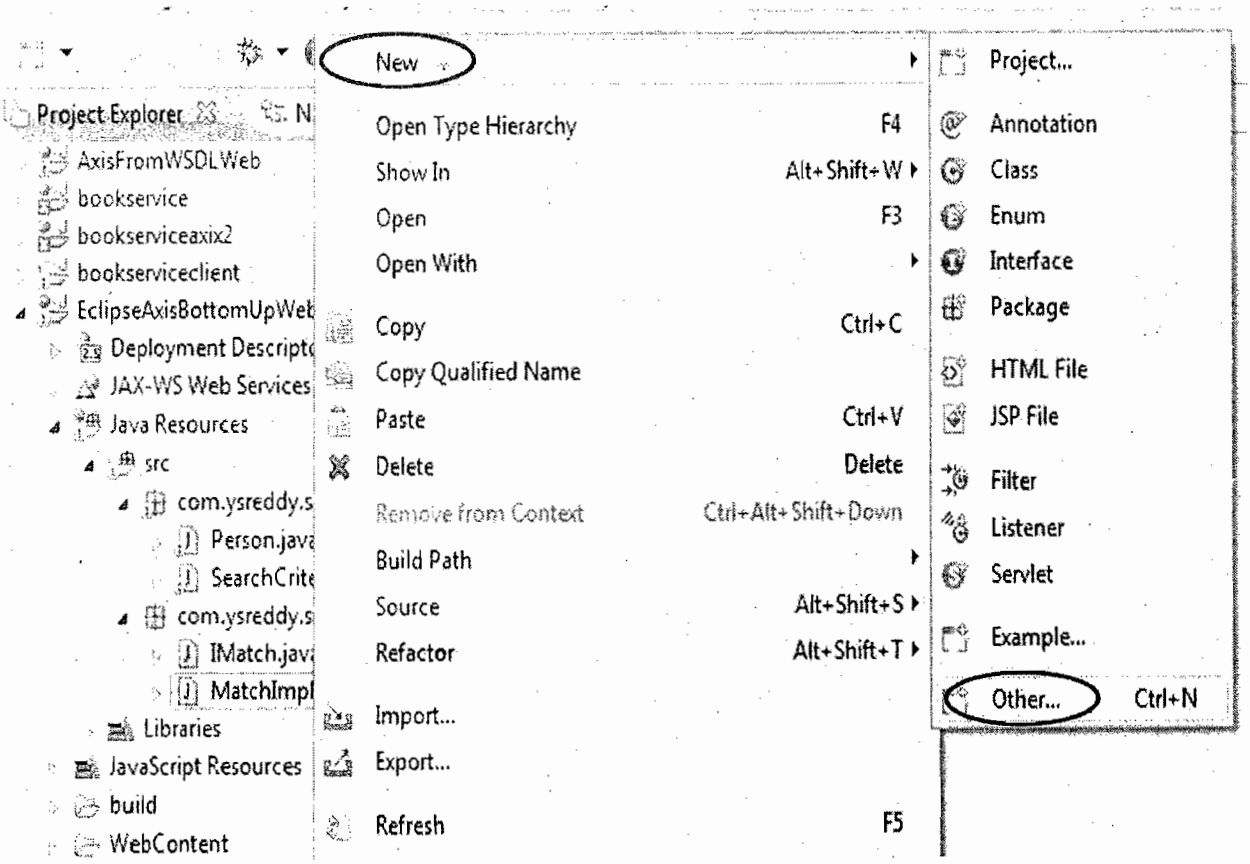
```
1. package com.ysreddy.shadi.service;
2.
3. import com.ysreddy.shadi.beans.Person;
4. import com.ysreddy.shadi.beans.SearchCriteria;
5.
6. public interface IMatch {
```

```
7.     public Person getRichMatch(SearchCriteria criteria);
8. }
```

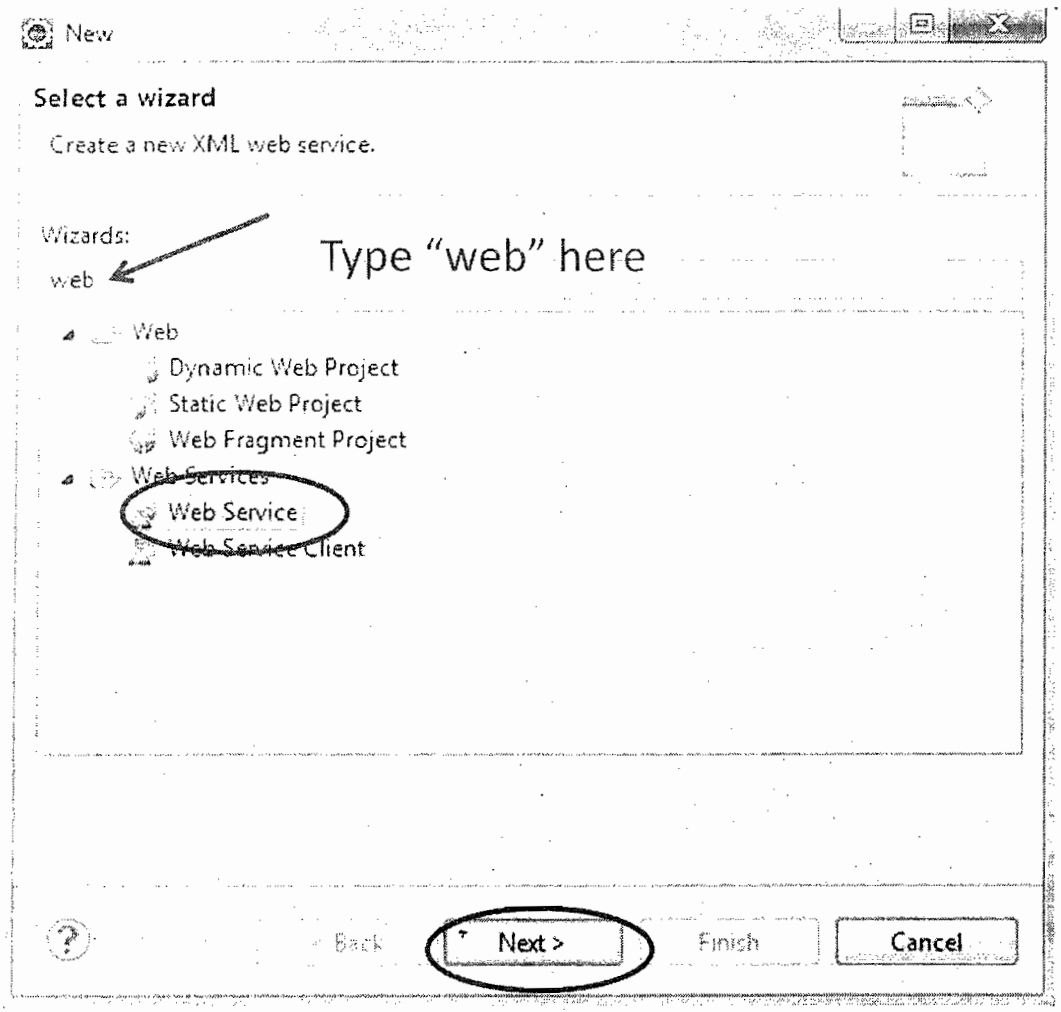
MatchImpl.java

```
1. IMatchpackage com.ysreddy.shadi.service;
2.
3. import com.ysreddy.shadi.beans.Person;
4. import com.ysreddy.shadi.beans.SearchCriteria;
5.
6. public class MatchImpl implements IMatch {
7.     public Person getRichMatch(SearchCriteria criteria) {
8.
9.           System.out.println(criteria.getAge());
10.                System.out.println(criteria.getSalary());
11.                System.out.println(criteria.getColor());
12.
13.                Person person = new Person();
14.                person.setName("ysreddy");
15.                person.setCity("Hyderbad");
16.
17.                return person;
18.           }
19.     }
```

Step 3: Right click on "**MatchImpl**" → New → Other



Step 4: Type "web" then select "**Web Service**"



Step 5: Select the following values

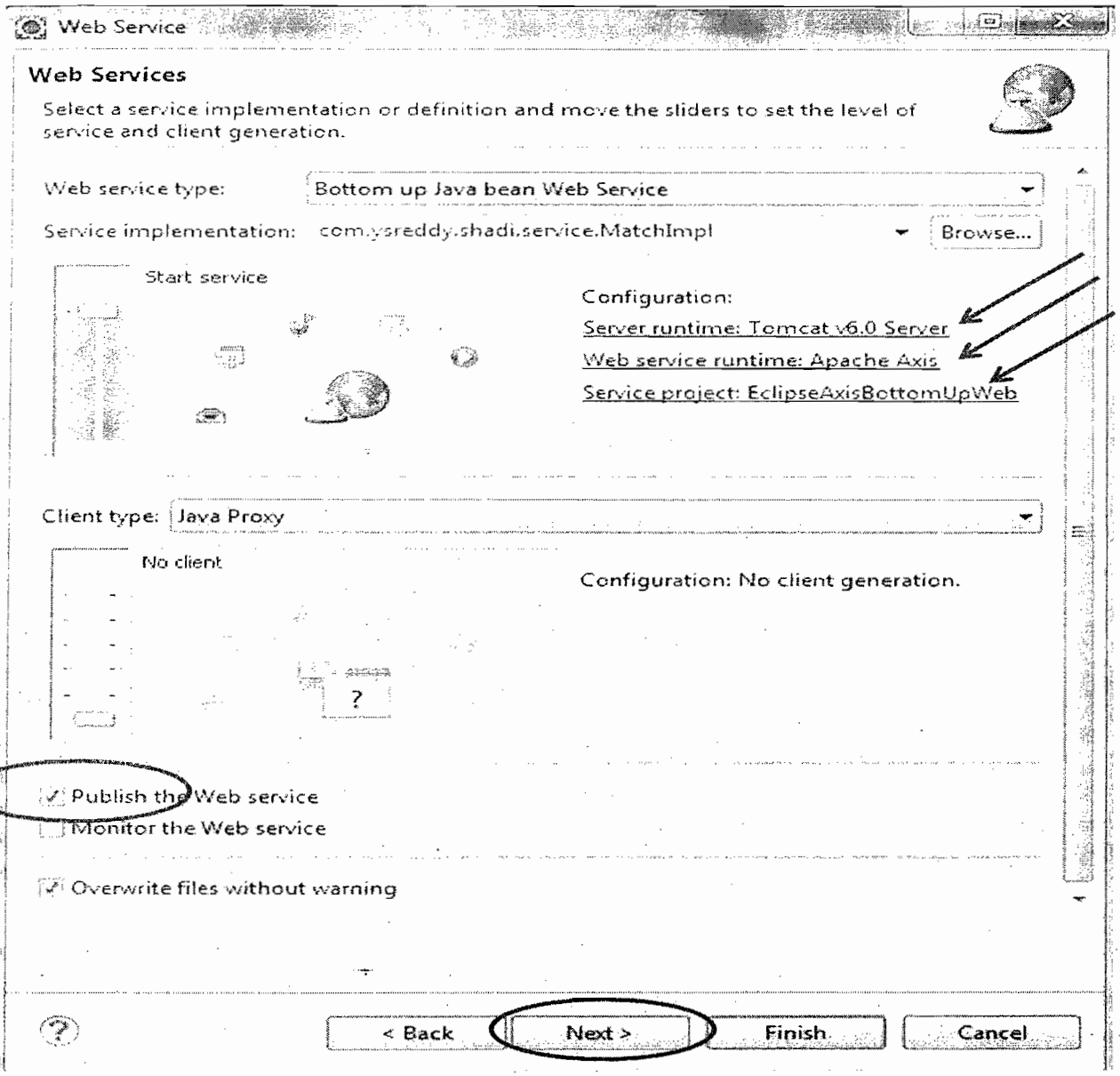
Server Runtime : "Tomcat v6.0 Server"

WebService Runtime: "Apache Axis"

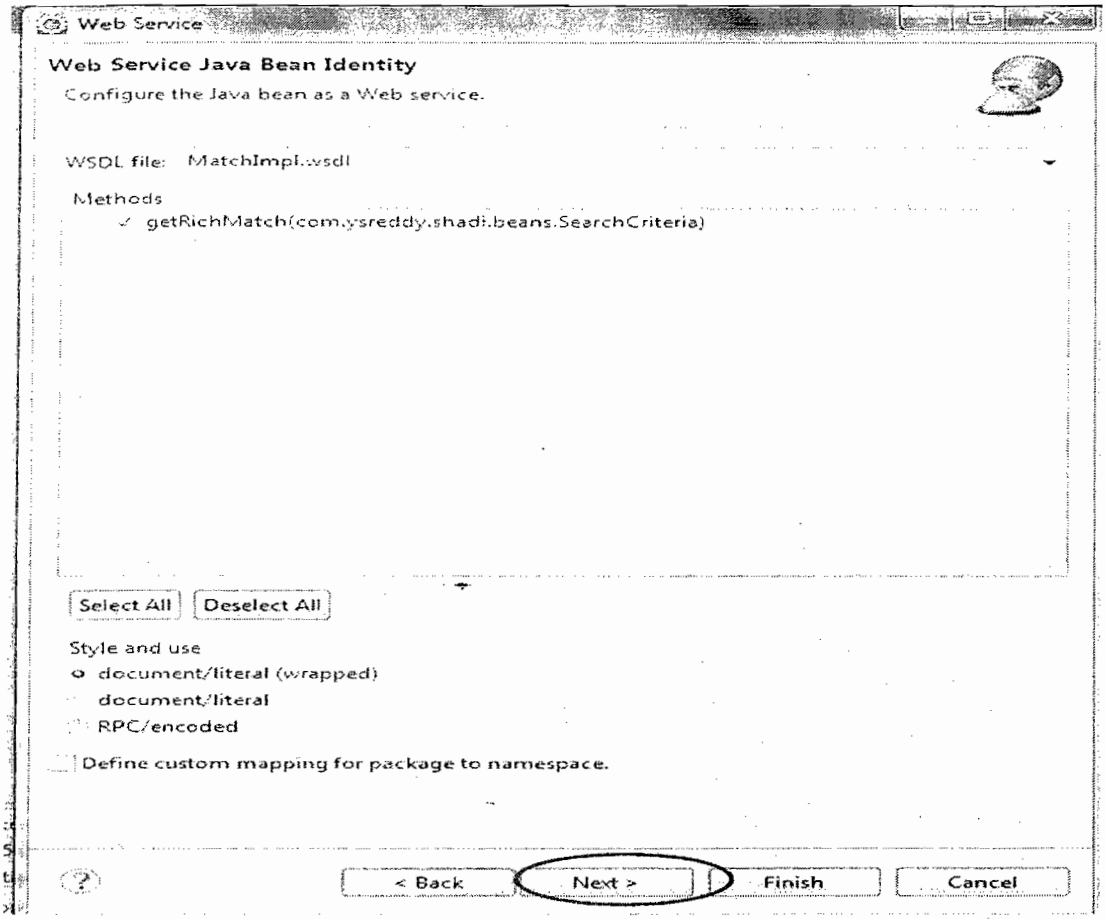
Service Project : "EclipseAxisBottomUpWeb"

Check the "Publish Web Service" check box

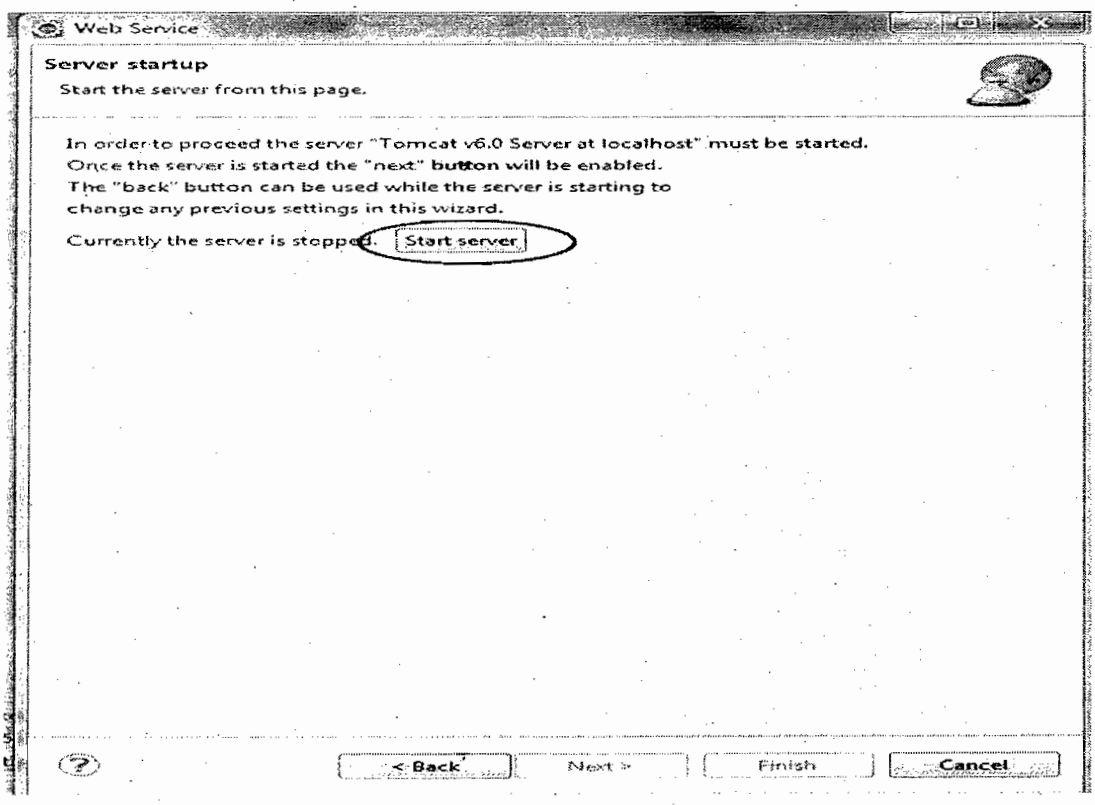
Now click on **"Next"**



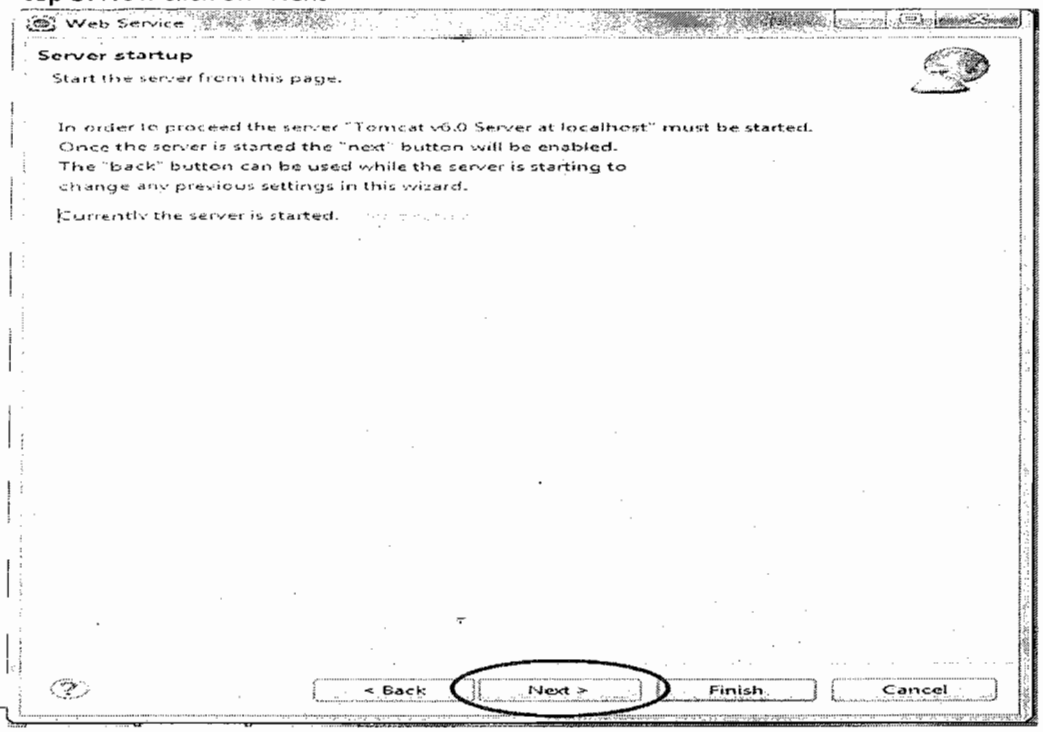
tep 6: Now click "Next"



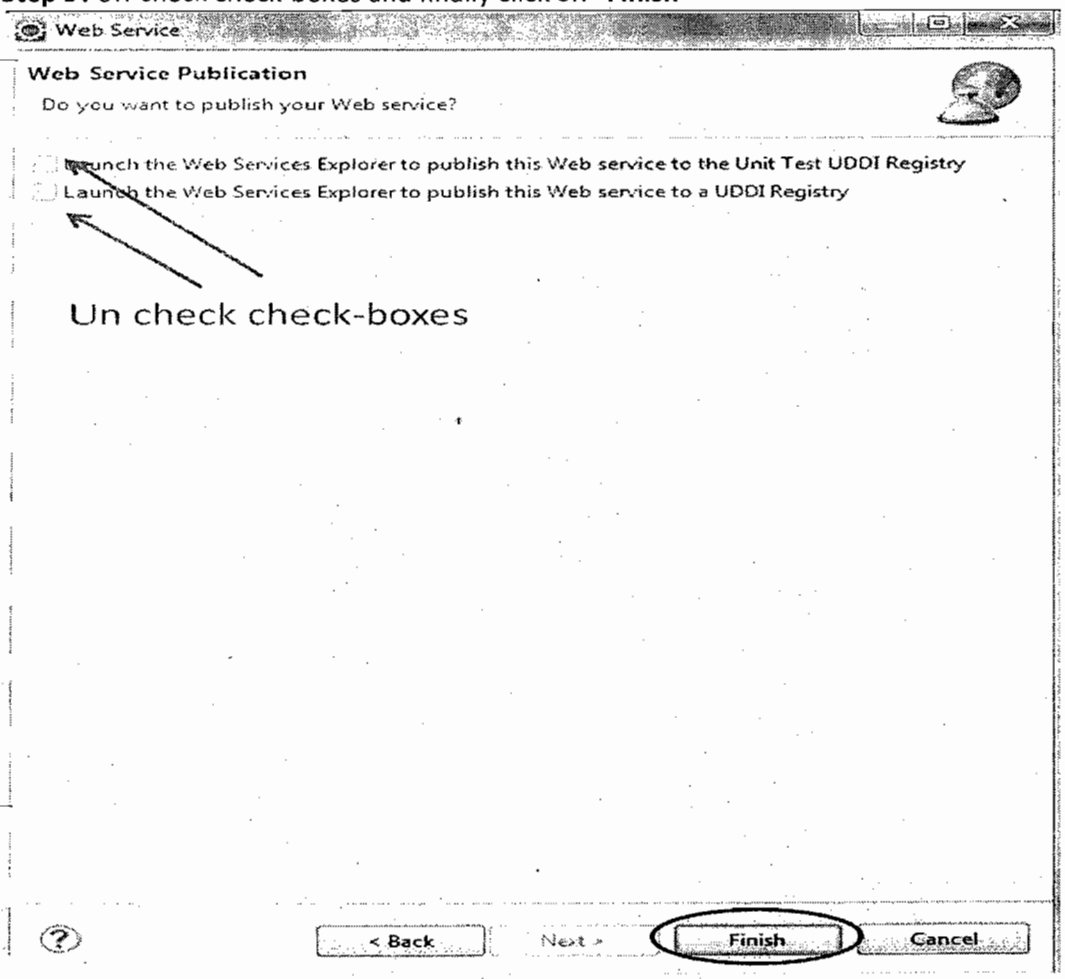
Step 7: Now click on **"start server"**



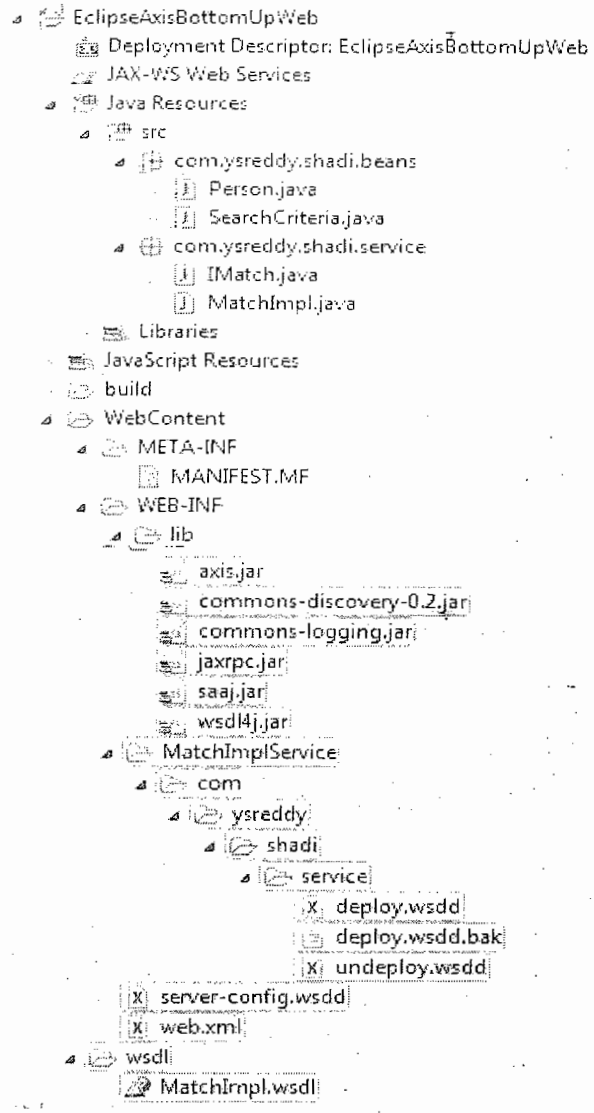
Step 8: Now click on "Next"



Step 9: Un-check check-boxes and finally click on **"Finish"**



Now you can find **server-config.wsdd, deploy.wsdd, undeploy.wsdd, MatchImpl.wsdl** files creation in our project. And Axis implementation jar files also added in the lib directory. Now the project structure is as follows...



web.xml

```
1. <?xml version="1.0" encoding="UTF-8"?>
2. <web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3.     xmlns="http://java.sun.com/xml/ns/javaee"
4. xmlns:web="http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
5.     xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
6. http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
7.     id="WebApp_ID" version="2.5">
8.     <display-name>EclipseAxisBottomUpWeb</display-name>
9.     <welcome-file-list>
10.                 <welcome-file>index.html</welcome-file>
11.                 <welcome-file>index.htm</welcome-file>
12.                 <welcome-file>index.jsp</welcome-file>
13.                 <welcome-file>default.html</welcome-file>
14.                 <welcome-file>default.htm</welcome-file>
15.                 <welcome-file>default.jsp</welcome-file>
16.     </welcome-file-list>
```

```
17.         <servlet>
18.               <display-name>Apache-Axis Servlet</display-name>
19.               <servlet-name>AxisServlet</servlet-name>
20.         <servlet-class>org.apache.axis.transport.http.AxisServlet</servlet-class>
21.         </servlet>
22.
23.         <servlet-mapping>
24.               <servlet-name>AxisServlet</servlet-name>
25.               <url-pattern>/servlet/AxisServlet</url-pattern>
26.         </servlet-mapping>
27.         <servlet-mapping>
28.               <servlet-name>AxisServlet</servlet-name>
29.               <url-pattern>*.jws</url-pattern>
30.         </servlet-mapping>
31.         <servlet-mapping>
32.               <servlet-name>AxisServlet</servlet-name>
33.               <url-pattern>/services/*</url-pattern>
34.         </servlet-mapping>
35.         <servlet>
36.               <display-name>Axis Admin Servlet</display-name>
37.               <servlet-name>AdminServlet</servlet-name>
38.         <servlet-class>org.apache.axis.transport.http.AdminServlet</servlet-class>
39.               <load-on-startup>100</load-on-startup>
40.         </servlet>
41.         <servlet-mapping>
42.               <servlet-name>AdminServlet</servlet-name>
43.               <url-pattern>/servlet/AdminServlet</url-pattern>
44.         </servlet-mapping>
45.   </web-app>
```

MatchImpl.wsdl

```
1. <?xml version="1.0" encoding="UTF-8"?>
2. <wsdl:definitions targetNamespace="http://service.shadi.ysreddy.com"
3.     xmlns:apachesoap="http://xml.apache.org/xml-soap"
4. xmlns:impl="http://service.shadi.ysreddy.com"
5.     xmlns:intf="http://service.shadi.ysreddy.com"
6. xmlns:tns1="http://beans.shadi.ysreddy.com"
7.     xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
8. xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
9.     xmlns:xsd="http://www.w3.org/2001/XMLSchema">
10.         <!--WSDL created by Apache Axis version: 1.4 Built on Apr 22, 2006 (06:55:48
11.               PDT) -->
12.         <wsdl:types>
13.               <schema elementFormDefault="qualified"
14.                targetNamespace="http://service.shadi.ysreddy.com"
15.                     xmlns="http://www.w3.org/2001/XMLSchema">
16.                     <import namespace="http://beans.shadi.ysreddy.com" />
17.                     <element name="getRichMatch">
18.                           <complexType>
19.                                 <sequence>
20.                           <element name="criteria" type="tns1:SearchCriteria" />
21.                                 </sequence>
22.                           </complexType>
23.                     </element>
24.                     <element name="getRichMatchResponse">
```

```
                        <complexType>
                              <sequence>
                        <element name="getRichMatchReturn" type="tns1:Person" />
                              </sequence>
                        </complexType>
                  </element>
            </schema>
            <schema elementFormDefault="qualified"
                targetNamespace="http://beans.shadi.ysreddy.com"
                  xmlns="http://www.w3.org/2001/XMLSchema">
                  <complexType name="SearchCriteria">
                        <sequence>
                              <element name="age" type="xsd:int" />
                              <element name="color" nillable="true"
                                                type="xsd:string" />
                              <element name="salary" type="xsd:double" />
                        </sequence>
                  </complexType>
                  <complexType name="Person">
                        <sequence>
                              <element name="city" nillable="true"
                                                type="xsd:string" />
                              <element name="name" nillable="true"
                                                type="xsd:string" />
                        </sequence>
                  </complexType>
            </schema>
      </wsdl:types>

      <wsdl:message name="getRichMatchRequest">

            <wsdl:part element="impl:getRichMatch" name="parameters">

            </wsdl:part>

      </wsdl:message>

      <wsdl:message name="getRichMatchResponse">

            <wsdl:part element="impl:getRichMatchResponse" name="parameters">

            </wsdl:part>

      </wsdl:message>

      <wsdl:portType name="MatchImpl">

            <wsdl:operation name="getRichMatch">

                  <wsdl:input message="impl:getRichMatchRequest"
                                                name="getRichMatchRequest">

                  </wsdl:input>

                  <wsdl:output message="impl:getRichMatchResponse"
```

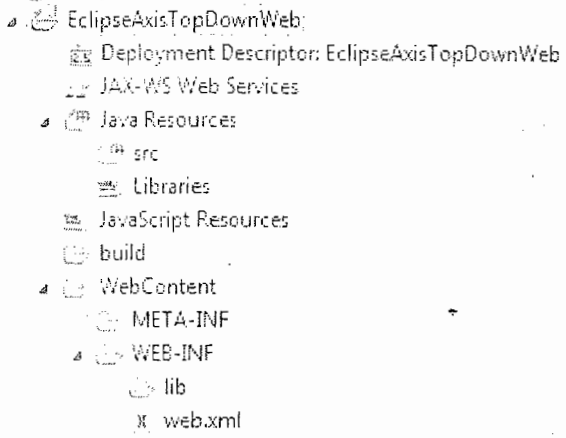
```
                                        name="getRichMatchResponse">

                    </wsdl:output>

              </wsdl:operation>

        </wsdl:portType>

        <wsdl:binding name="MatchImplSoapBinding" type="impl:MatchImpl">

              <wsdlsoap:binding style="document"
                    transport="http://schemas.xmlsoap.org/soap/http" />

              <wsdl:operation name="getRichMatch">

                    <wsdlsoap:operation soapAction="" />

                    <wsdl:input name="getRichMatchRequest">

                          <wsdlsoap:body use="literal" />

                    </wsdl:input>

                    <wsdl:output name="getRichMatchResponse">

                          <wsdlsoap:body use="literal" />

                    </wsdl:output>

              </wsdl:operation>

        </wsdl:binding>

        <wsdl:service name="MatchImplService">

              <wsdl:port binding="impl:MatchImplSoapBinding" name="MatchImpl">

                    <wsdlsoap:address

location="http://localhost:8086/EclipseAxisBottomUpWeb/services/MatchImpl" />

              </wsdl:port>

        </wsdl:service>

</wsdl:definitions>
```

Step 10: Open the browser and send the request to server with the following url, then you can get the wsdl of our web service

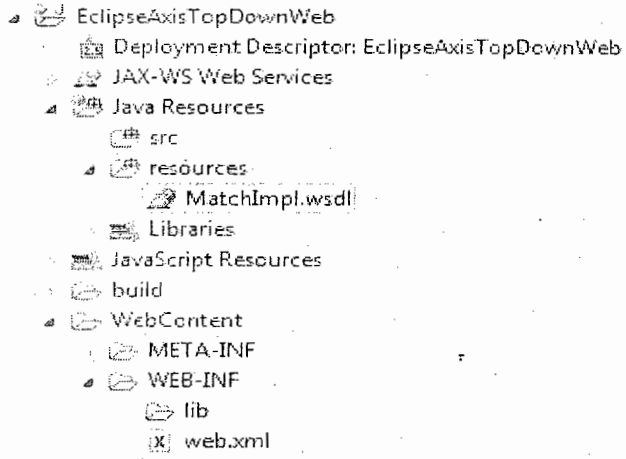
http://localhost:8086/EclipseAxisBottomUpWeb/services/MatchImpl?wsdl

Q.) Develop Web service using top-down approach, with eclipse IDE(JAX-RPC API, AXIS IMPLEMENTATION)?

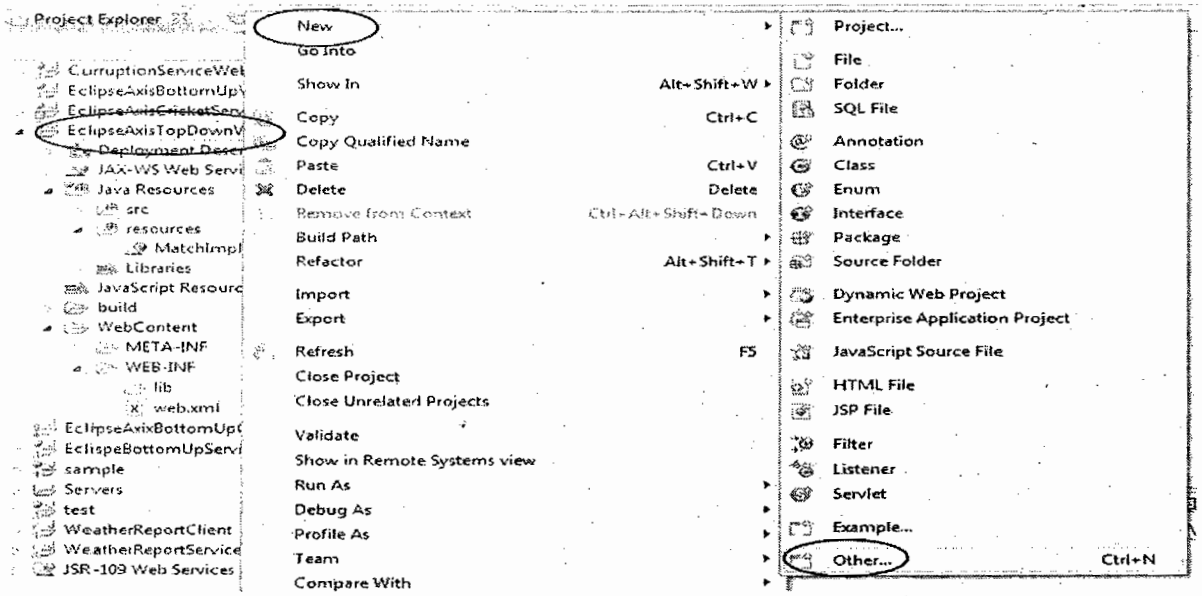
Step 1: Create a **"Dynamic Web Project"** with some name like **"EclipseAxisTopDownWeb"**



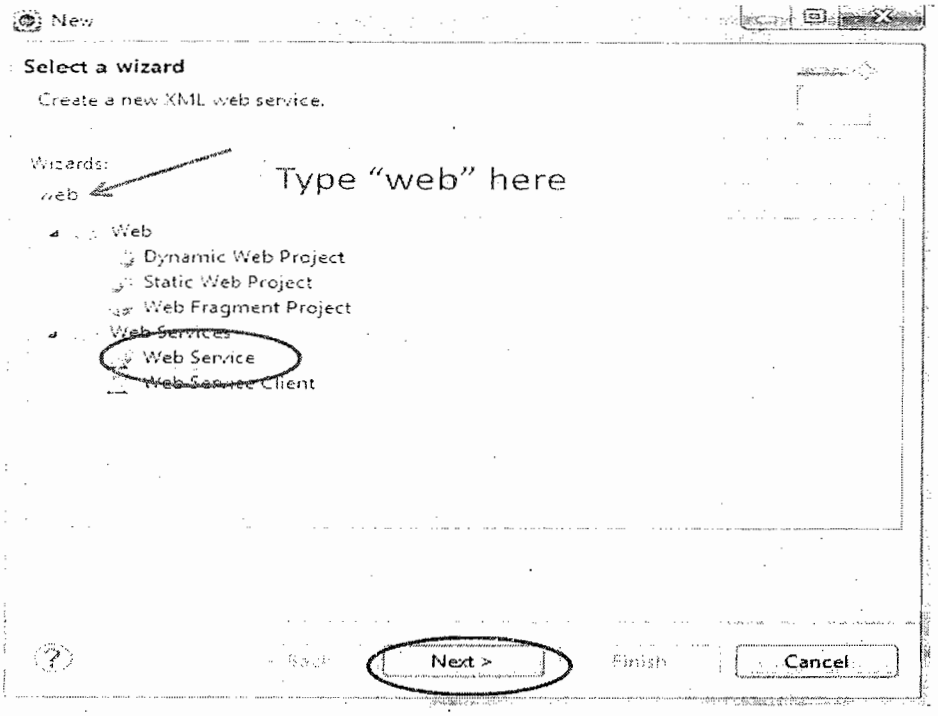
Step 2: create one source folder with name **"resources"** , And place WSDL document(which was generated in the previous application) in the **"resources"** folder.



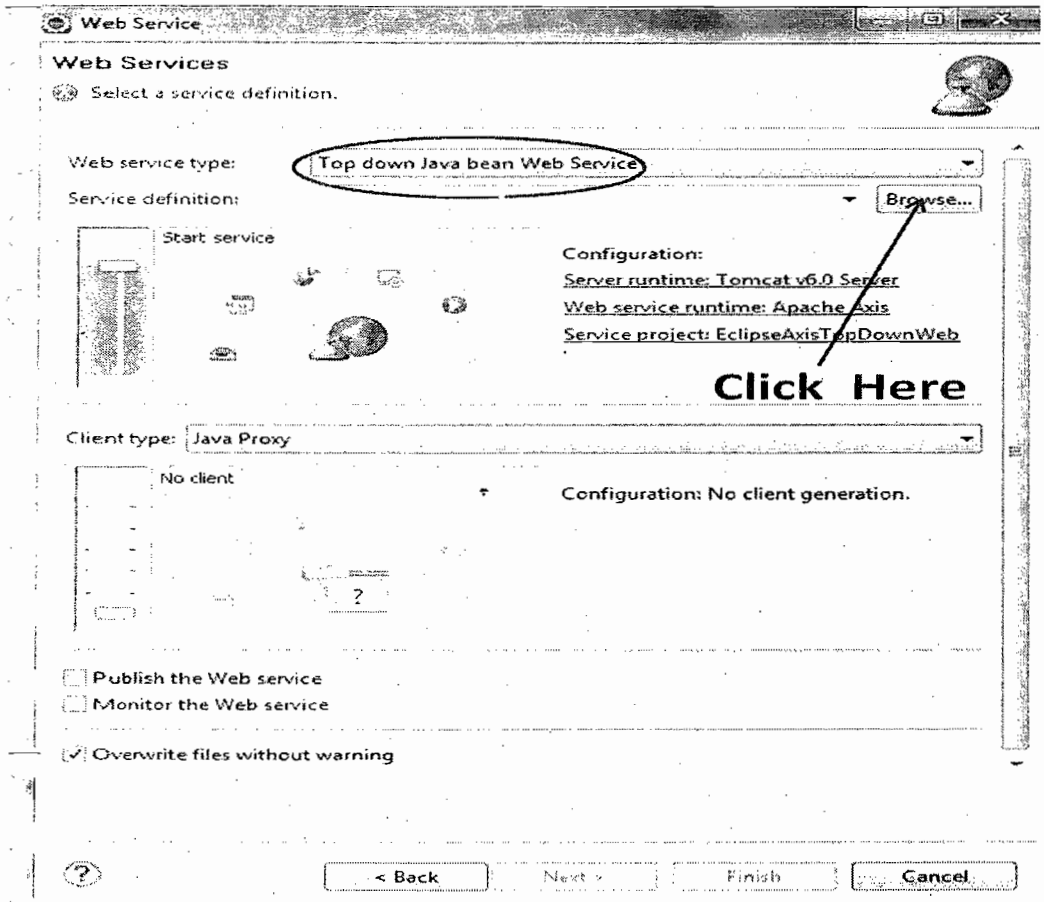
Step 3: Right click on the project → New → Other



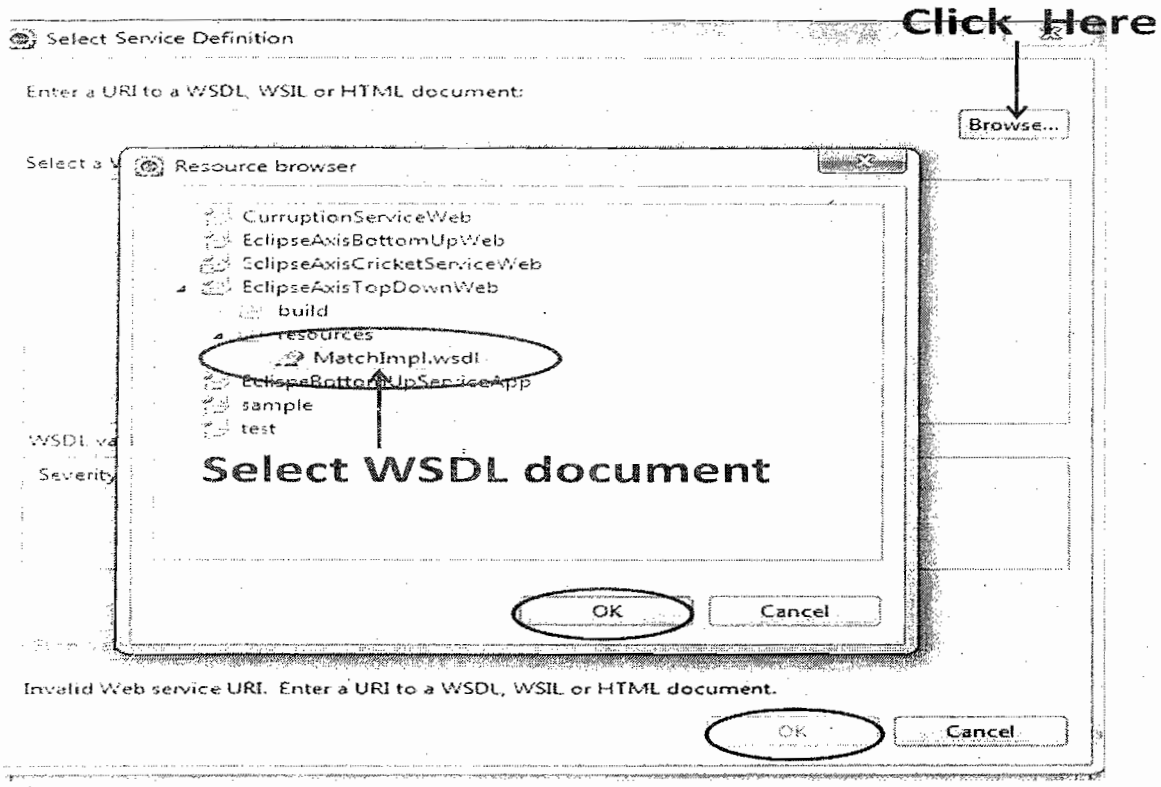
Step 4: Type "web" then select "**Web Service**"



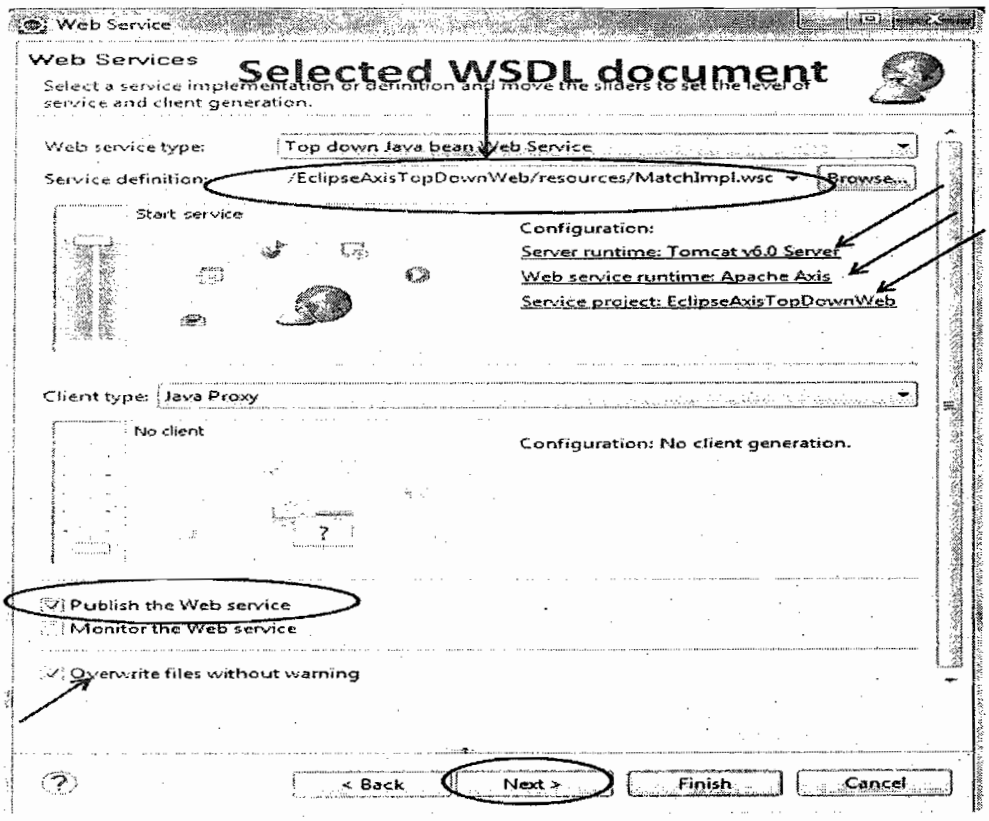
Step 5: Select the following values



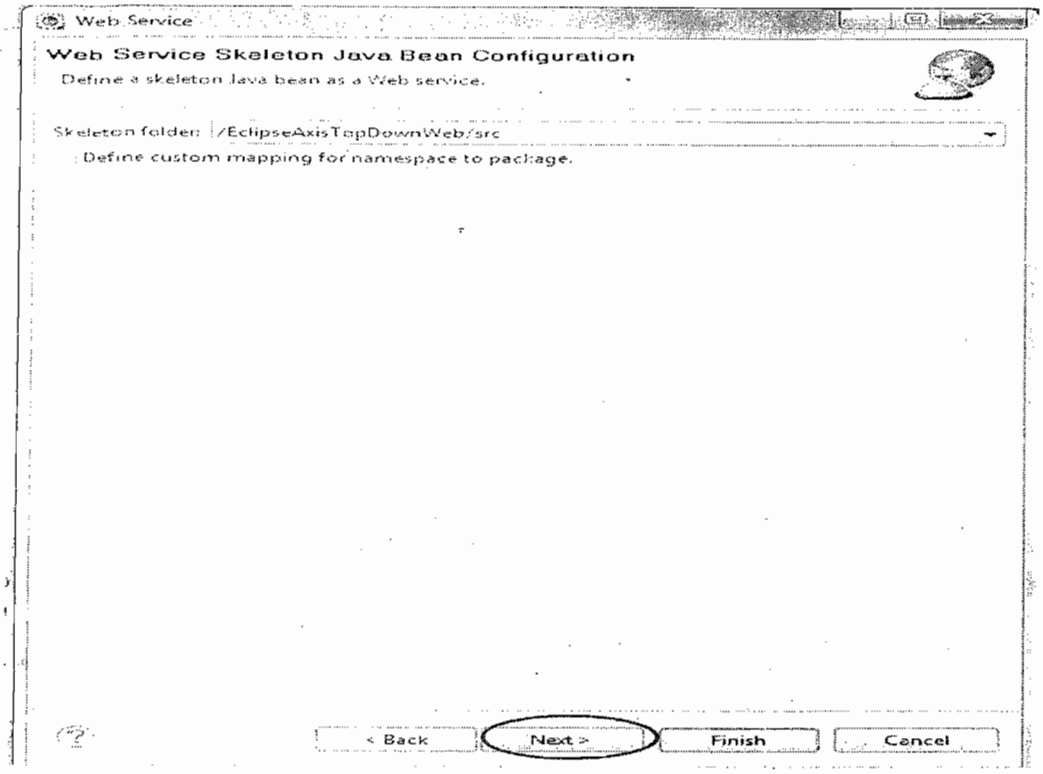
Step 5: Then you will get one pop-up, then select the following values



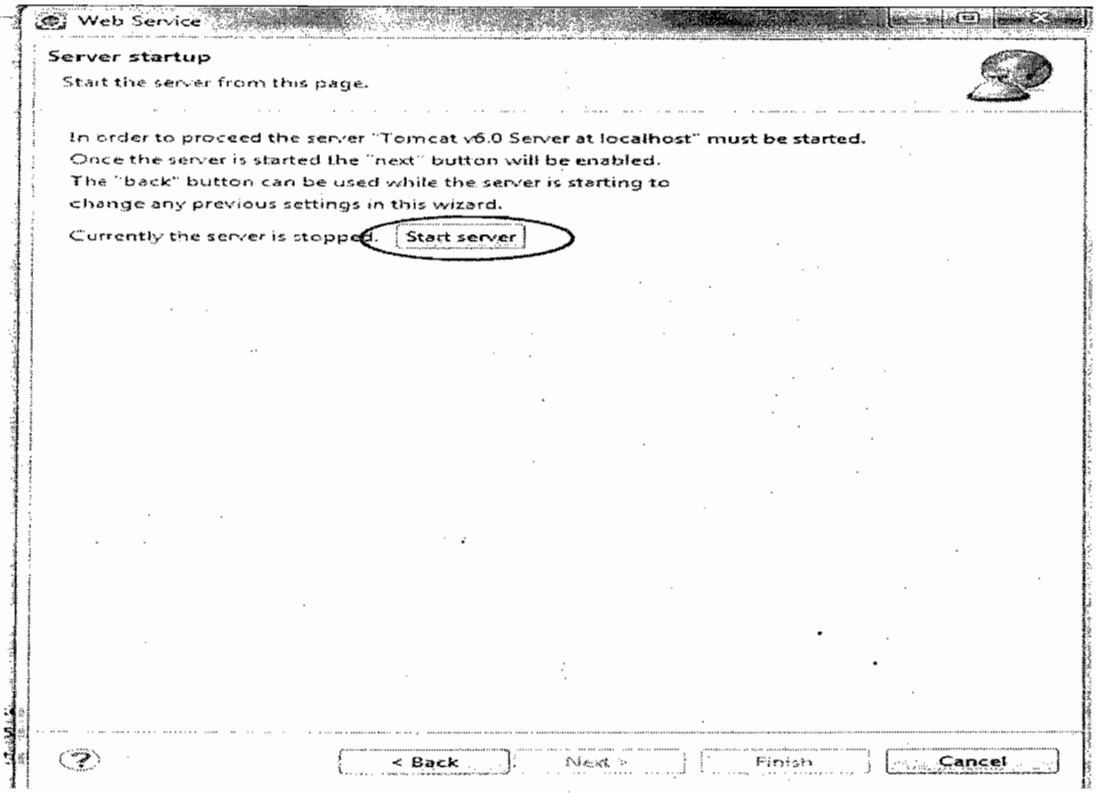
Step 6: Then select the following values, Then click on "Next"



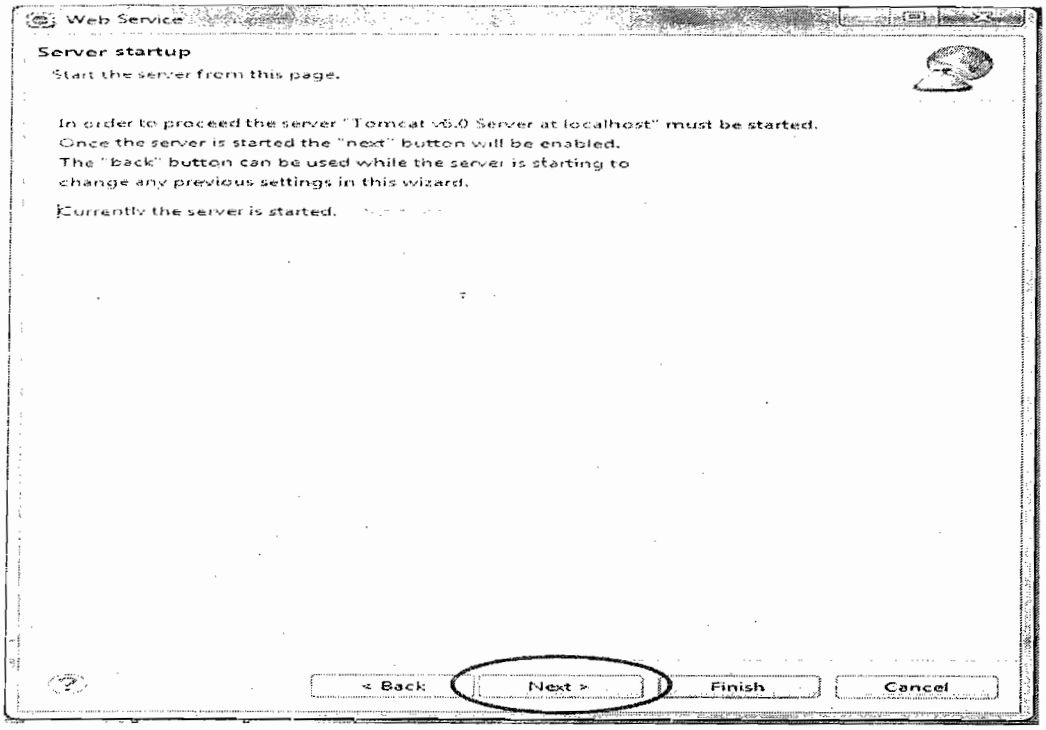
Step 7: Then click on "Next"



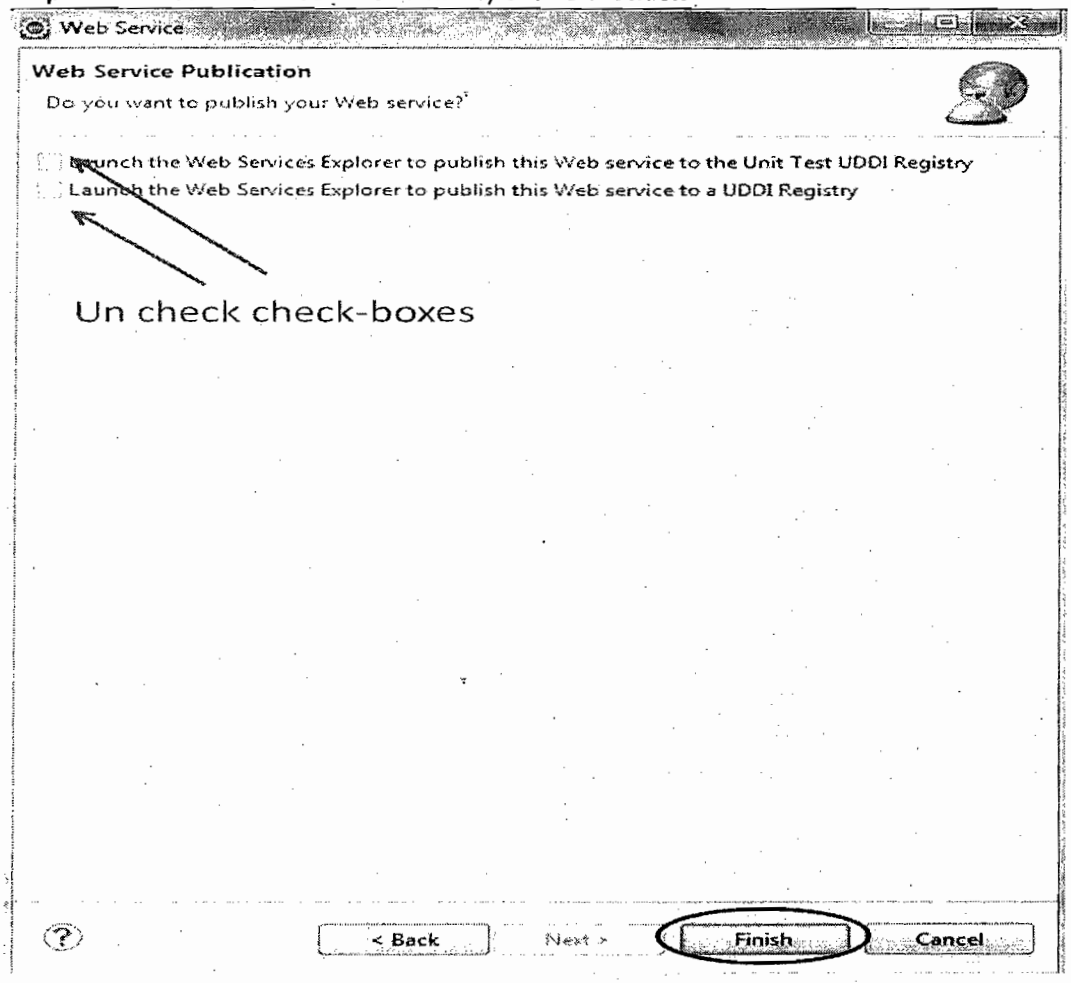
Step 8: Now click on **"start server"**



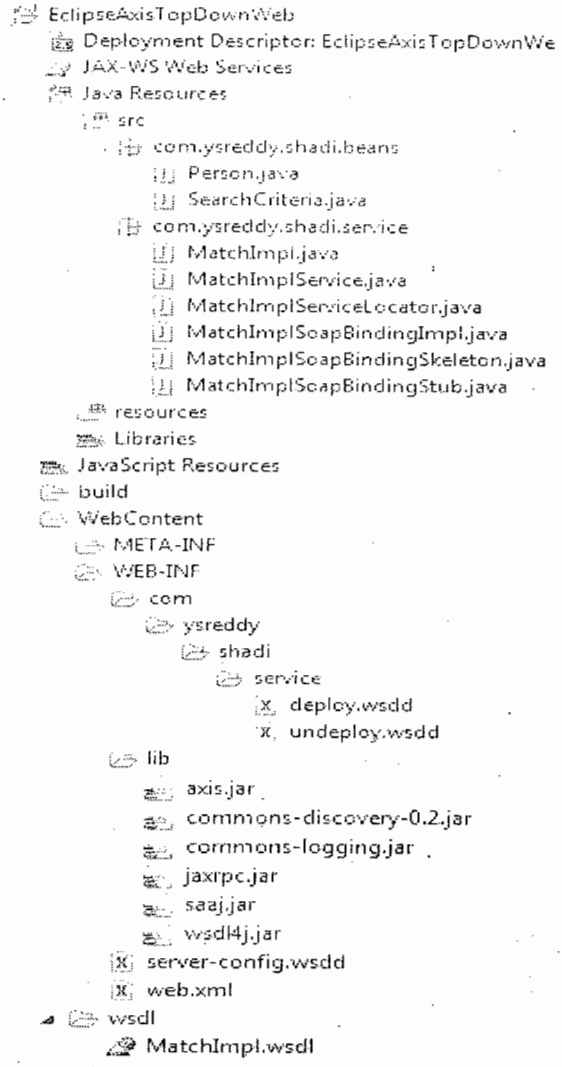
Step 8: Now click on "Next"



Step 9: Un-check check-boxes and finally click on **"Finish"**



Now you can find **server-config.wsdd, deploy.wsdd, undeploy.wsdd, MatchImpl.wsdl** files creation in our project. And Axis implementation jar files also added in the lib directory. Now the project structure is as follows...



Step 10: Now You can find one class with name "**MatchImplSoapBindingImpl**", In this class we need to implement our business logic, then again redeploy our application into server.

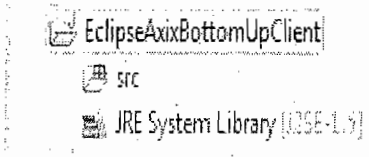
MatchImplSoapBindingImpl.java

```
1. /**
2.  * MatchImplSoapBindingImpl.java
3.  *
4.  * This file was auto-generated from WSDL
5.  * by the Apache Axis 1.4 Apr 22, 2006 (06:55:48 PDT) WSDL2Java emitter.
6.  */
7.
8. package com.ysreddy.shadi.service;
9.
10.    import com.ysreddy.shadi.beans.Person;
11.
12.    public class MatchImplSoapBindingImpl implements
   com.ysreddy.shadi.service.MatchImpl{
13.        public com.ysreddy.shadi.beans.Person
```

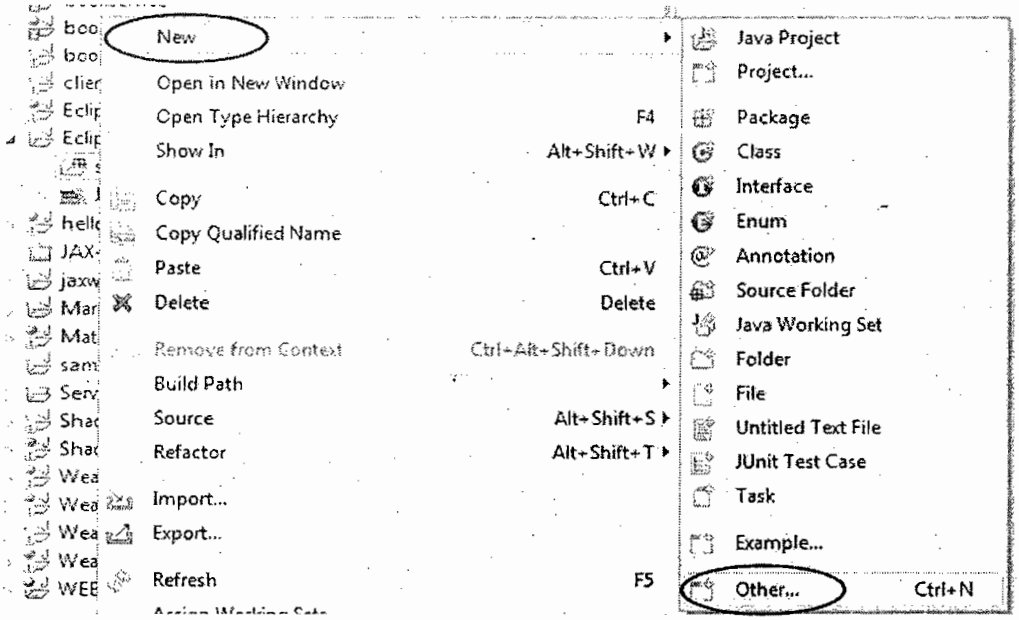
```
14.   getRichMatch(com.ysreddy.shadi.beans.SearchCriteria criteria) throws
15.                                                    java.rmi.RemoteException {
16.               System.out.println(criteria.getAge());
17.               System.out.println(criteria.getSalary());
18.               System.out.println(criteria.getColor());
19.
20.               Person person = new Person();
21.               person.setName("ysreddy");
22.               person.setCity("Hyderbad");
23.
24.               return person;
25.
26.       }
27.
28.   }
```

Q.) Develop Web service client using Axis, with eclipse IDE?

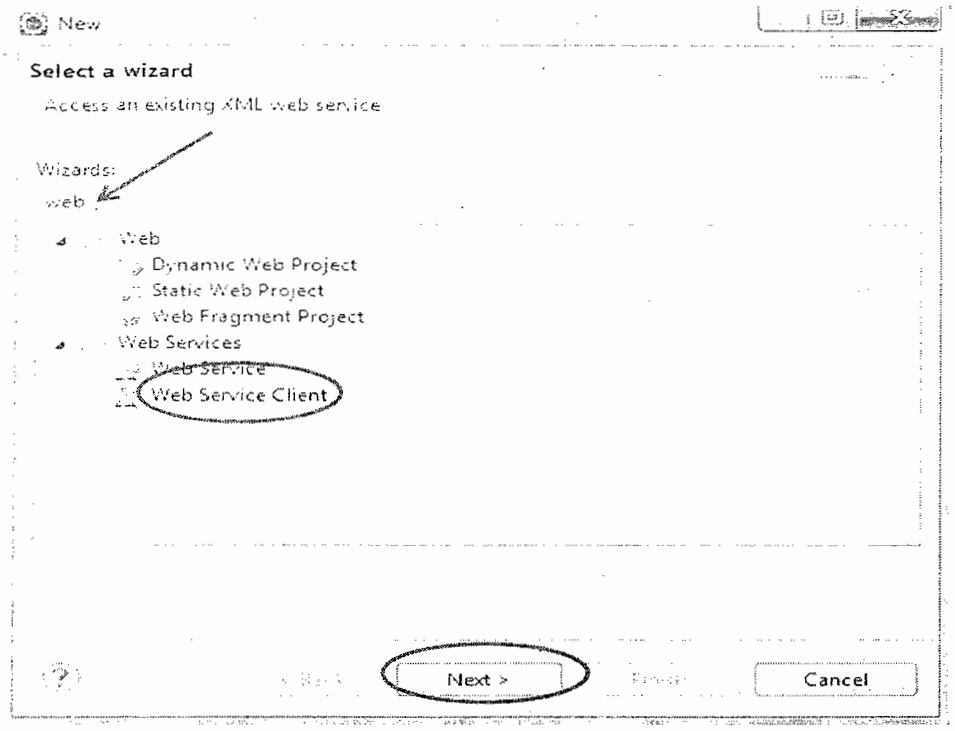
Step 1: create a project with name **"EclipseAxixBottomUpClient"**



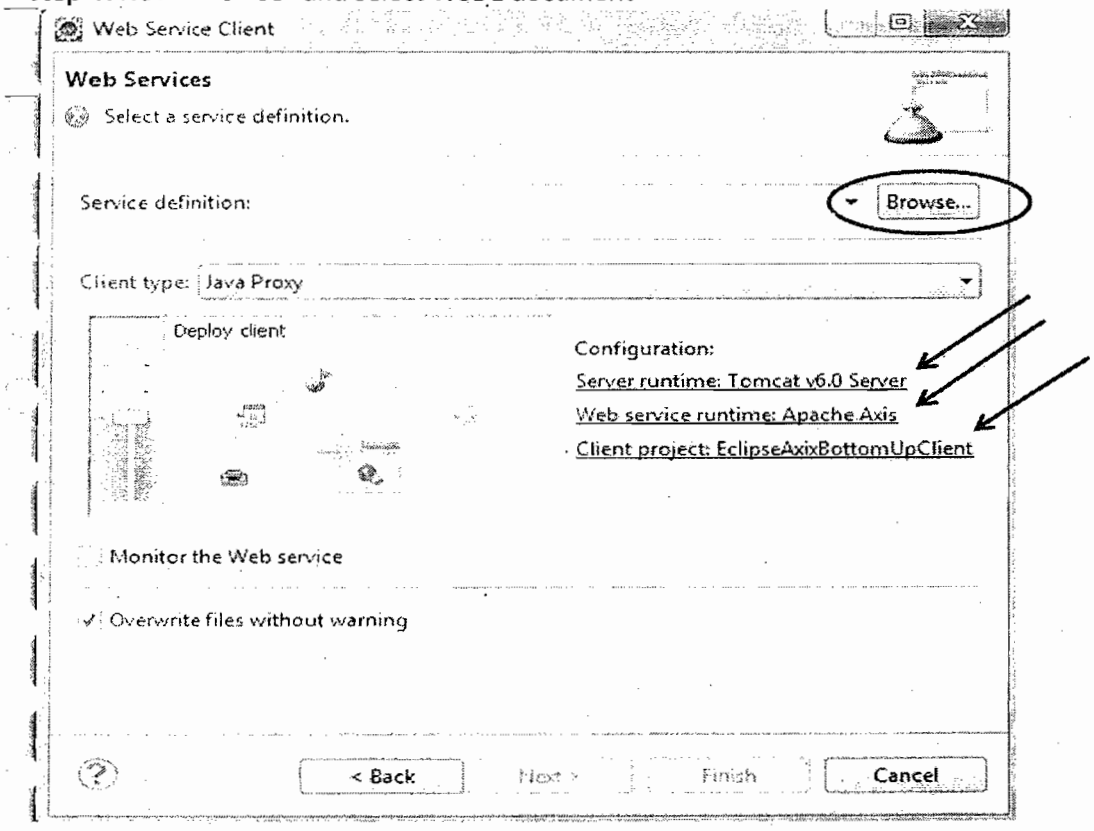
Step 2: Rich click on the "src" folder → New → Other



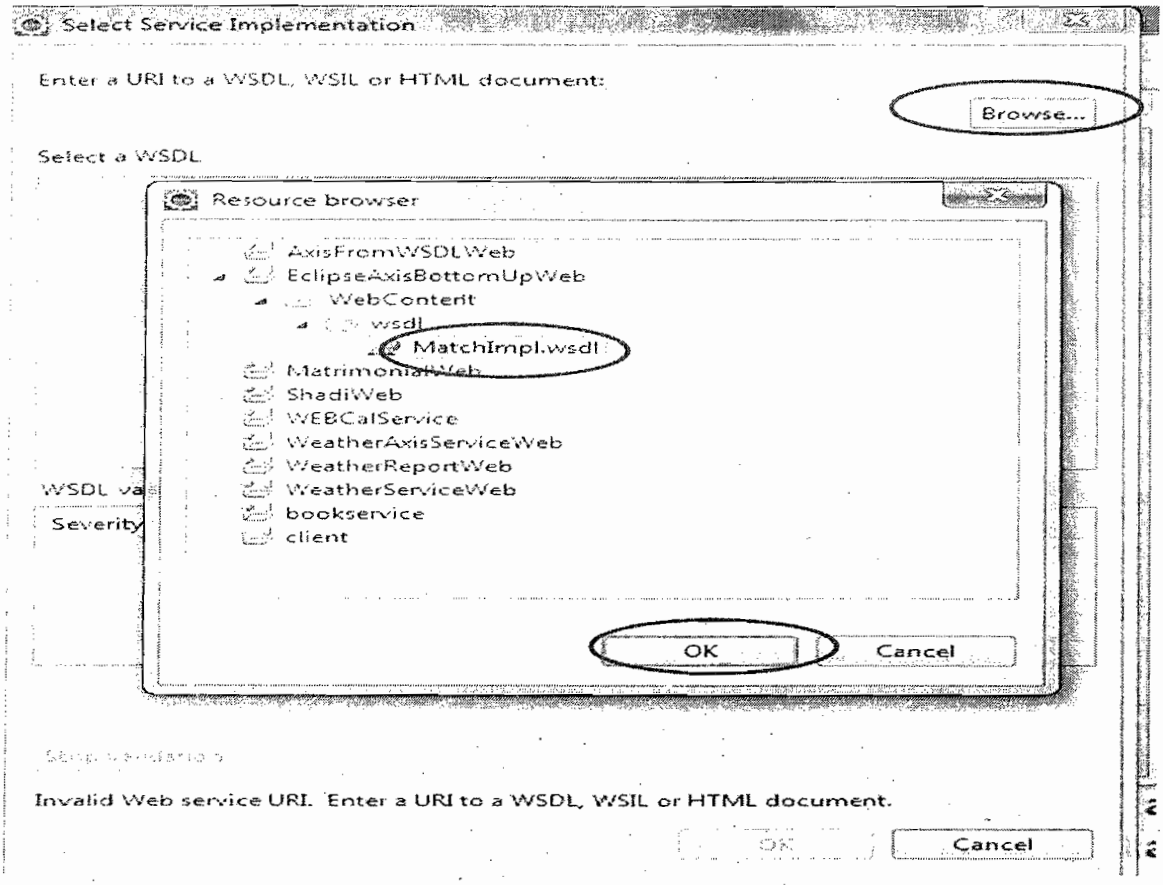
Step 3: Type "web" then select "**Web Service Client",** Then click on "Next" button



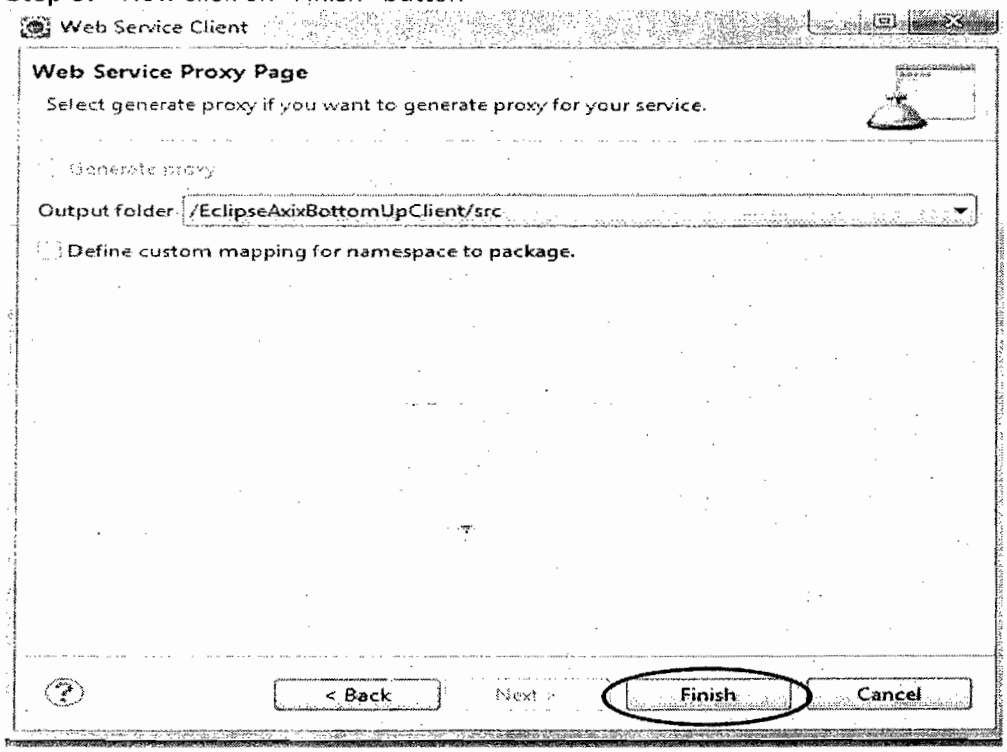
Step 4: Now "Browse" and select WSDL document



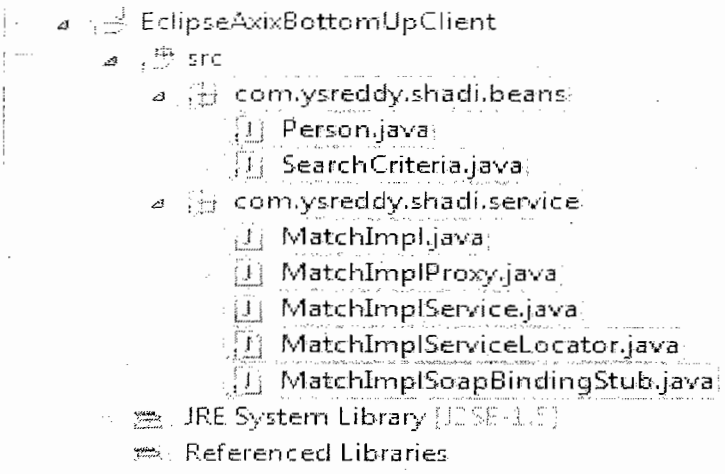
Step 5: Select WSDL documents, and then click "Next"



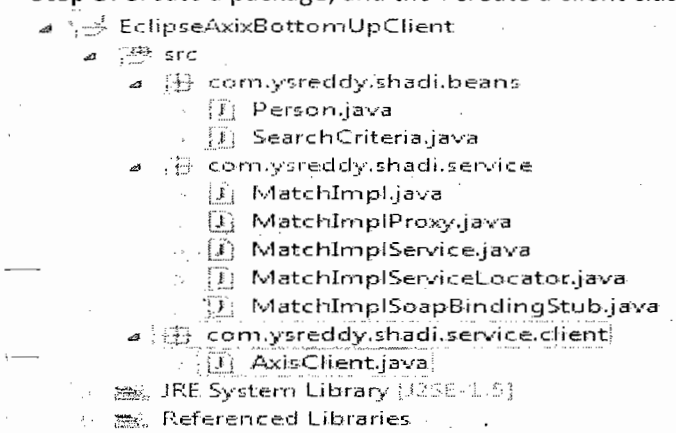
Step 6: Now click on "Finish" button



Step 7: Then the following stubs will be generated.



Step 8: Create a package, and then create a client class



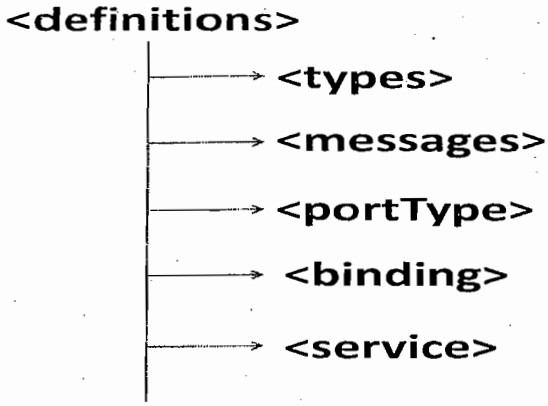
AxisClient.java

```
1. package com.ysreddy.shadi.service.client;
2.
3. import java.rmi.RemoteException;
4. import javax.xml.rpc.ServiceException;
5. import com.ysreddy.shadi.beans.Person;
6. import com.ysreddy.shadi.beans.SearchCriteria;
7. import com.ysreddy.shadi.service.MatchImpl;
8. import com.ysreddy.shadi.service.MatchImplServiceLocator;
9. public class AxisClient {
10.         public static void main(String[] args) throws ServiceException,
11.                     RemoteException {
12.               MatchImplServiceLocator locator = new MatchImplServiceLocator();
13.               MatchImpl match = locator.getMatchImpl();
14.               SearchCriteria criteria = new SearchCriteria();
15.               criteria.setAge(26);
16.               criteria.setColor("fair");
17.               criteria.setSalary(9879);
18.
19.               Person person = match.getRichMatch(criteria);
20.               System.out.println("Better match for u....");
21.               System.out.println("Person name : " + person.getName());
22.               System.out.println("Person City : " + person.getCity());
23.         }
24.   }
```

WSDL

- WSDL stands for Web Services Description Language
- WSDL is pronounced as 'wiz-dull' and spelled out as 'W-S-D-L'
- WSDL is an XML document, which is validated against http://schemas.xmlsoap.org/wsdl/ namespace
- WSDL versions are 1.1, 2.0
- WSDL is a W3C recommendation
- The WSDL describe about web service such as **interface name, Method names, Parameter types, Return type, Message format(such as SOAP), Network protocol (such as HTTP), and Endpoint address(URI).**
- WSDL can be used to generate skeletons(server side artifacts), stubs(client side artifacts) using code generation tools(like wscompile, wsgen...etc)

Basic Structure of WSDL



<**definitions**> : Root element of WSDL

<types> : Defines Input and output Data Types, Exception types(using XML Schema)

<message> : Defines the data elements of an operation i.e. Input message and output message of the operation

< portType> : Represents Service Endpoint Interface(SEI), describes the kinds of operations that a Web service provides

<binding> : Defines the message (such as SOAP format), Transport protocol (such as HTTP, SMTP, etc), Messaging style (such as RPC or Document), Encoding styles(such as 'literal' or soap encoding)

<wsdl:service> : Assign an Endpoint address (URI) to a specific port

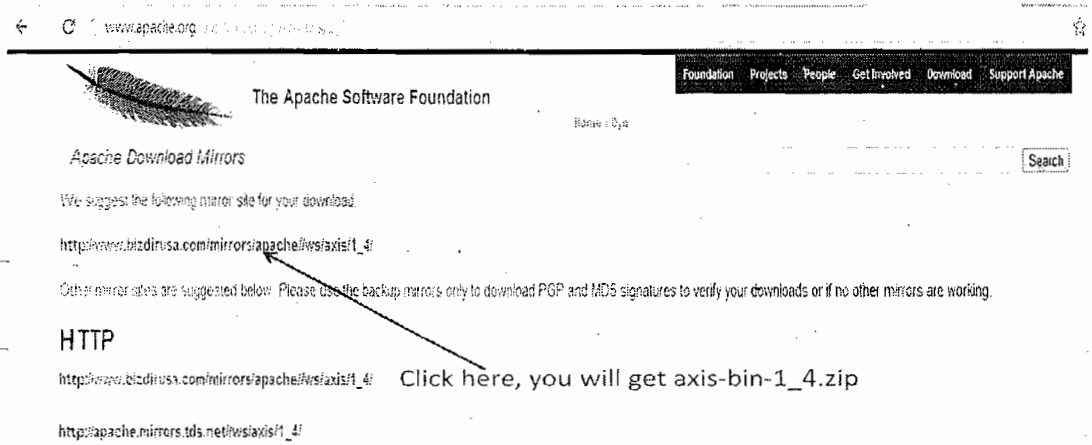
JAX-RPC API, Apache Axis Implementation, Contract first approach

This includes the following steps

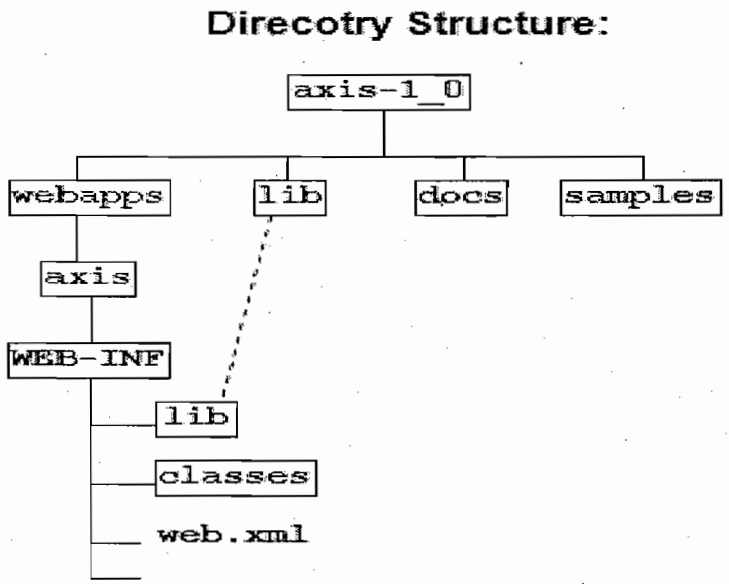
1. Axis Project Setup
2. Axis Environment setup To Run command line tools
3. Web service development with Axis(contract first approach)
4. Deploy our web service into Web server

1.Axis Project Setup

Step 1: Download axis-1_4.zip binary distribution zip file from the following link -> http://www.apache.org/dyn/closer.cgi/ws/axis/1_4/, you can use any of the mirrors available.

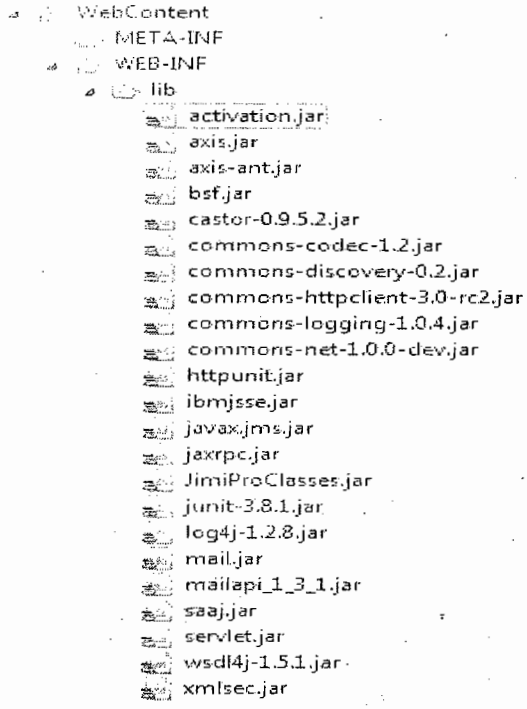


Step 2: Extract the zip file and copy the entire **axis-1_4** folder to **c:** then you can find the following directory structure

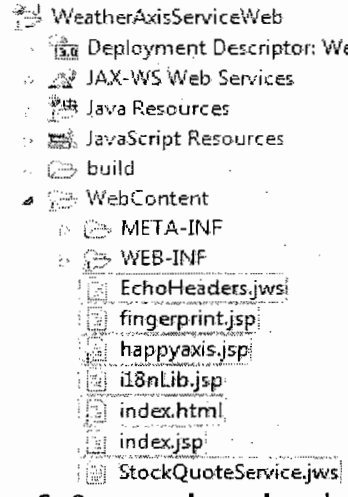


Step 3: create Dynamic Web Project with name "**WeatherAxisServiceWeb**"

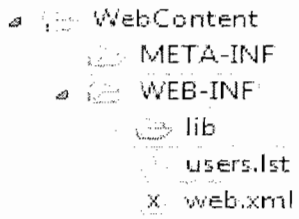
Step 4: copy the required jar files **into WebContent/WEB-INF/lib**



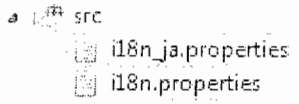
Step 5: Copy all ***.html, *.jsp and *.jws** files from **C:\axis-1_4\webapps\axis** to **WeatherAxisServiceWeb\WebContent**



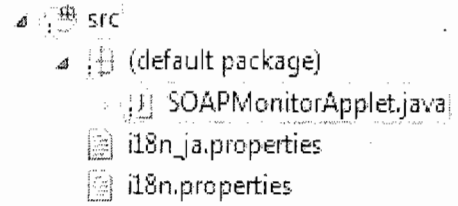
Step 6: Copy **web.xml** and **users.lst** files from **C:\axis-1_4\webapps\axis\WEB-INF** to **WeatherAxisServiceWeb\WebContent\WEB-INF**



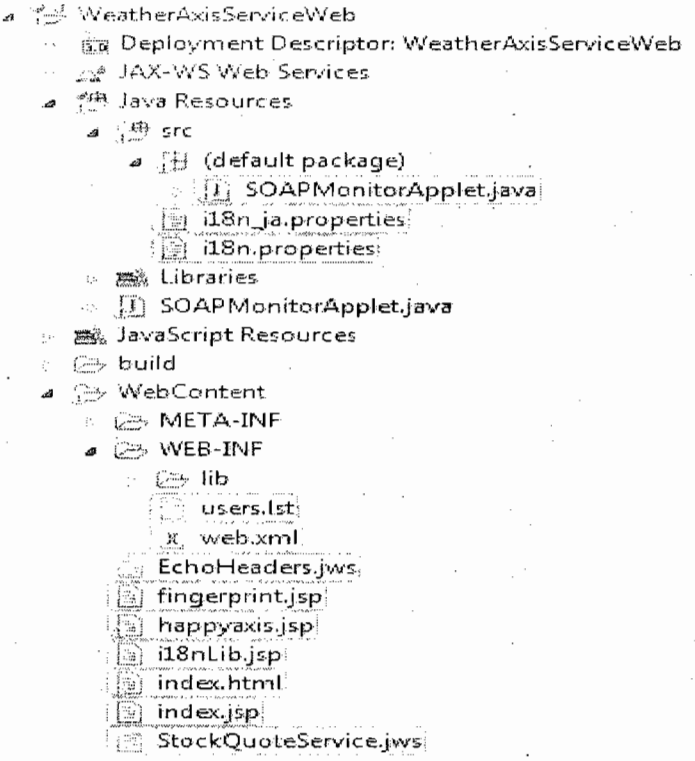
Step 7: Copy **i18n.properties and i18n_ja.properties from C:\axis-1_4\webapps\axis\WEB-INF\classes** to **"src"** of our project.



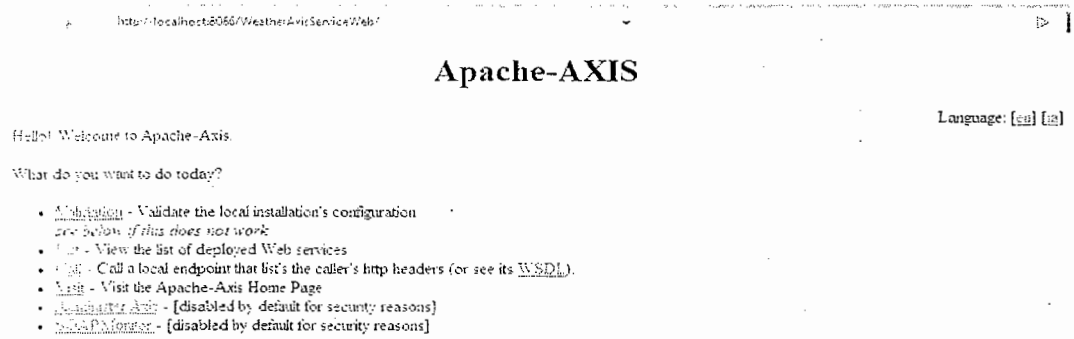
Step 8: Copy *.java from **C:\axis-1_4\webapps\axis** to "**src**" of our project



After all the above steps our project structure should like as follows...



Step 9: Now deploy our project onto Web Server and start your server. Access your application with **http://localhost:8086/WeatherAxisServiceWeb/** This should show you happy axis home page.

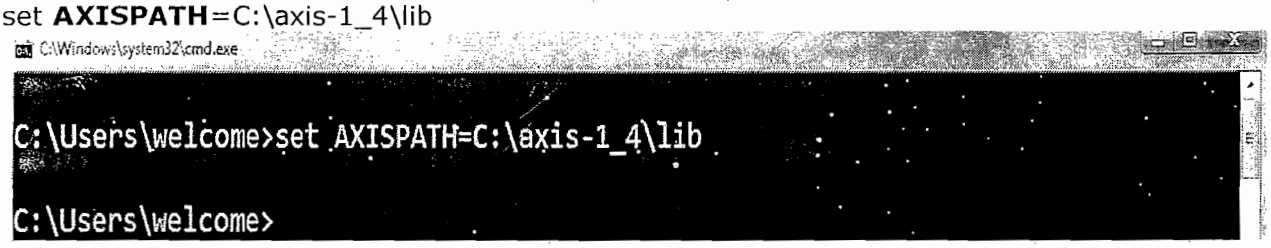


validate To check your project dependencies, if we are missing any jar files it will intimate missing dependencies.

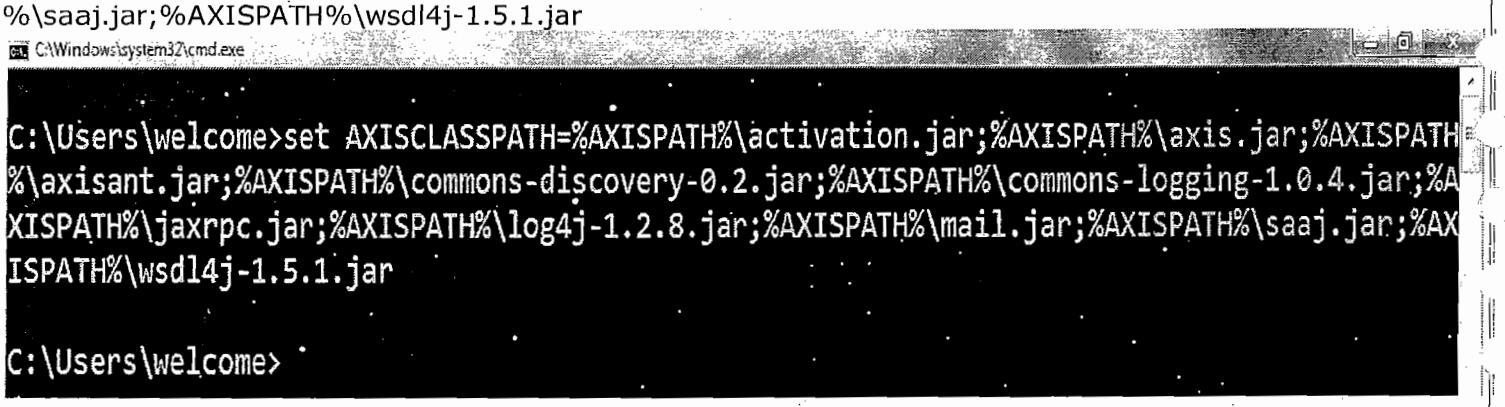
2.Axis Environment setup To Run command line tools

Step 1: Set the **AXISPATH** to the Axis libraries that are under **c:\axis-1_4\lib** directory as below

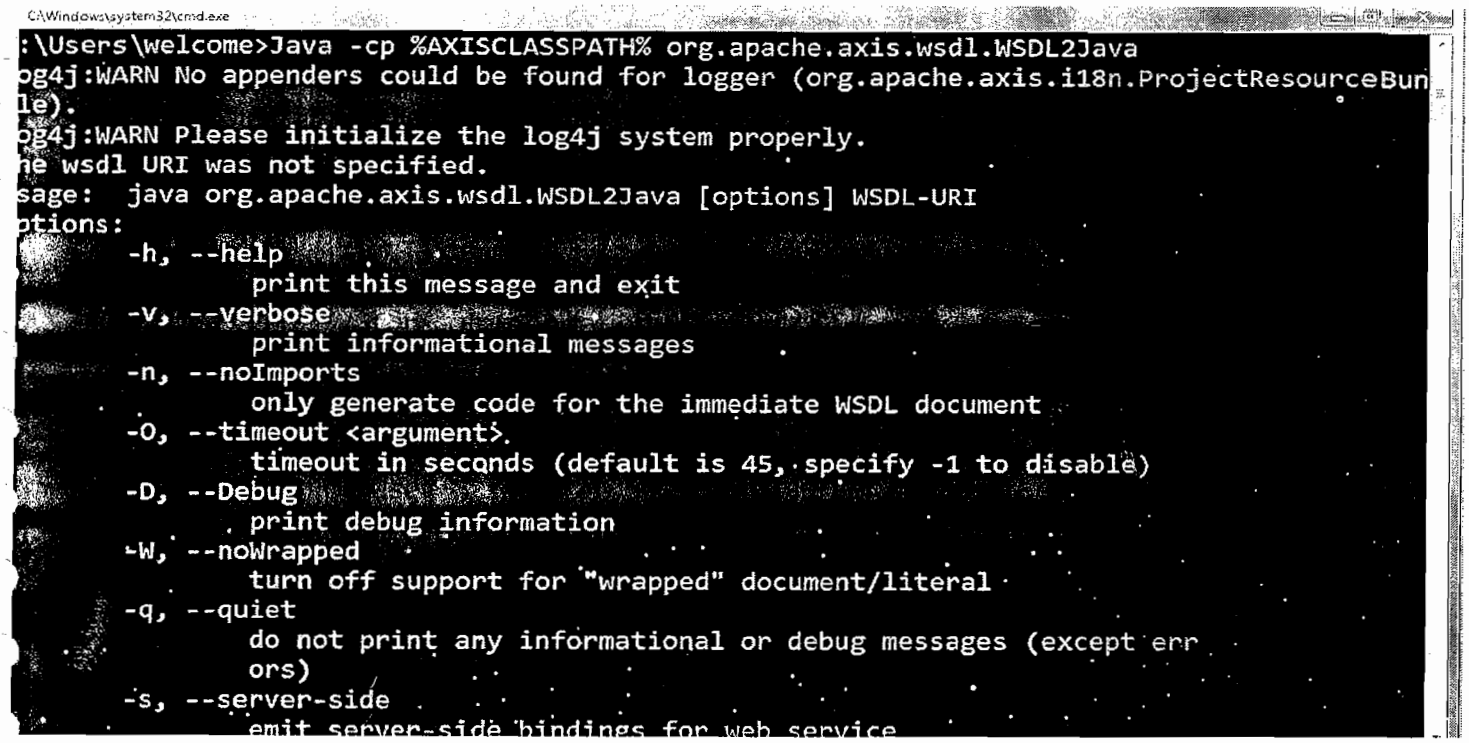
set **AXISPATH**=C:\axis-1_4\lib



set **AXISCLASSPATH**=%AXISPATH%\activation.jar;%AXISPATH%\axis.jar;%AXISPATH%\axisant.jar;%AXISPATH%\commons-discovery-0.2.jar;%AXISPATH%\commons-logging-1.0.4.jar;%AXISPATH%\jaxrpc.jar;%AXISPATH%\log4j-1.2.8.jar;%AXISPATH%\mail.jar;%AXISPATH%\saaj.jar;%AXISPATH%\wsdl4j-1.5.1.jar



Step 2: Now type the following command
Java -cp %AXISCLASSPATH% org.apache.axis.wsdl.WSDL2Java
If show set of switches as follows, then our setup is perfect, otherwise re-check setup again…



3.Web service development with Axis(contract first approach)

Step 1: create a WSDL Document under **WebContent/WEB-INF** directory **<WeatherService.wsdl>**

WeatherService.wsdl

```
1.  <definitions xmlns="http://schemas.xmlsoap.org/wsdl/"
2.     xmlns:tns="http://ysreddy.com/weather/wsdl"
3.  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
4.     xmlns:ns2="http://ysreddy.com/weather/types"
5.  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
6.     name="WeatherService" targetNamespace="http://ysreddy.com/weather/wsdl">
7.
8.  <types>
9.           <schema xmlns="http://www.w3.org/2001/XMLSchema"
10.                  xmlns:tns="http://ysreddy.com/weather/types"
11.                  xmlns:soap11-enc="http://schemas.xmlsoap.org/soap/encoding/"
12.                  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
13.                  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
14.                  targetNamespace="http://ysreddy.com/weather/types">
15.                  <import namespace="http://schemas.xmlsoap.org/soap/encoding/" />
16.
17.                  <complexType name="WeatherCriteria">
18.                        <sequence>
19.                              <element name="city" type="string" />
20.                              <element name="country" type="string" />
21.                              <element name="state" type="string" />
22.                        </sequence>
23.                  </complexType>
24.                  <complexType name="WeatherStatus">
25.                        <sequence>
26.                              <element name="description" type="string" />
```

```
27.                                 <element name="humidity" type="double" />
28.                                 <element name="temperature" type="int" />
29.                             </sequence>
30.                         </complexType>
31.                 </schema>
32.         </types>
33.
34.         <message name="IWeatherInfo_getWeatherInfo">
35.                 <part name="WeatherCriteria_1" type="ns2:WeatherCriteria" />
36.         </message>
37.         <message name="IWeatherInfo_getWeatherInfoResponse">
38.                 <part name="result" type="ns2:WeatherStatus" />
39.         </message>
40.
41.         <portType name="IWeatherInfo">
42.                 <operation name="getWeatherInfo" parameterOrder="WeatherCriteria_1">
43.                         <input message="tns:IWeatherInfo_getWeatherInfo" />
44.                         <output message="tns:IWeatherInfo_getWeatherInfoResponse" />
45.                 </operation>
46.         </portType>
47.
48.         <binding name="IWeatherInfoBinding" type="tns:IWeatherInfo">
49.                 <soap:binding transport="http://schemas.xmlsoap.org/soap/http"
50.                         style="rpc" />
51.                 <operation name="getWeatherInfo">
52.                         <soap:operation soapAction="" />
53.                         <input>
54.                           <soap:body
55.                             encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
56.                             use="encoded" namespace="http://ysreddy.com/weather/wsdl" />
57.                         </input>
58.                         <output>
59.                            <soap:body
60.                              encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
61.                              use="encoded" namespace="http://ysreddy.com/weather/wsdl" />
62.                         </output>
63.                 </operation>
64.         </binding>
65.         <service name="WeatherService">
66.                 <port name="IWeatherInfoPort" binding="tns:IWeatherInfoBinding">
67.                 <soap:address xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
68.                   location="http://localhost:8086/WeatherServiceWeb/ws/IWeatherInfo" />
69.                 </port>
70.         </service>
71.   </definitions>
```

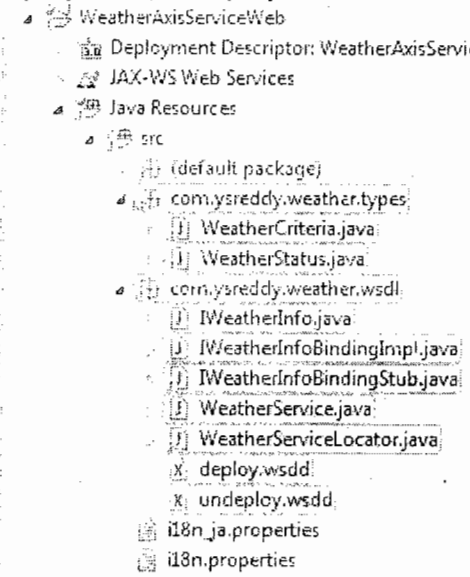
Step 2: Now navigate to the project root directory **E:\ws\WeatherAxisServiceWeb**

Step 3: Type the following command

E:\ws\WeatherAxisServiceWeb> Java -cp %AXISCLASSPATH% org.apache.axis.wsdl.WSDL2Java -v --server-side --skeletonDeploy false -o src WebContent\WEB-INF\WeatherService.wsdl

```
Mark C:\Windows\system32\cmd.exe
E:\ws\WeatherAxisServiceWeb>Java -cp %AXISCLASSPATH% org.apache.axis.wsdl.WSDL2Java -v --s
erver-side --skeletonDeploy false -o src WebContent\WEB-INF\WeatherService.wsdl
log4j:WARN No appenders could be found for logger (org.apache.axis.i18n.ProjectResourceBun
dle).
log4j:WARN Please initialize the log4j system properly.
Parsing XML file:  WebContent\WEB-INF\WeatherService.wsdl
Generating src\com\ysreddy\weather\types\WeatherCriteria.java
Generating src\com\ysreddy\weather\types\WeatherStatus.java
Generating src\com\ysreddy\weather\wsdl\IWeatherInfo.java
Generating src\com\ysreddy\weather\wsdl\IWeatherInfoBindingStub.java
Generating src\com\ysreddy\weather\wsdl\IWeatherInfoBindingImpl.java
Generating src\com\ysreddy\weather\wsdl\WeatherService.java
Generating src\com\ysreddy\weather\wsdl\WeatherServiceLocator.java
Generating src\com\ysreddy\weather\wsdl\deploy.wsdd
Generating src\com\ysreddy\weather\wsdl\undeploy.wsdd

E:\ws\WeatherAxisServiceWeb>
```

Step 3: Now go to project and refresh the project, you can find SEI interface, Implementation, input and output objects, **deploy.wsdd, undeploy.wsdd** files under "**src**" folder.



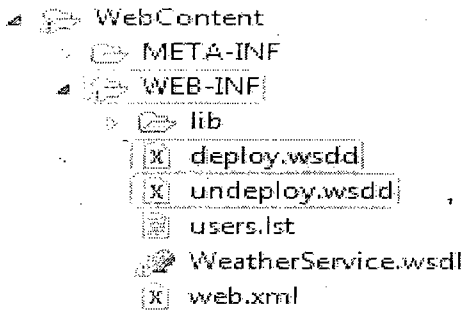
Step 4: Now implement business logic in Service Implementation class.

_WeatherInfoBindingImpl.java

```
1. /**
2.  * IWeatherInfoBindingImpl.java
3.  *
4.  * This file was auto-generated from WSDL
5.  * by the Apache Axis 1.4 Apr 22, 2006 (06:55:48 PDT) WSDL2Java emitter.
6.  */
7.
8. package com.ysreddy.weather.wsdl;
9.
10.    import com.ysreddy.weather.types.WeatherStatus;
11.
12.    public class IWeatherInfoBindingImpl implements
13.                                    com.ysreddy.weather.wsdl.IWeatherInfo{
```

```
14.   public com.ysreddy.weather.types.WeatherStatus
15.   getWeatherInfo(com.ysreddy.weather.types.WeatherCriteria weatherCriteria) throws
16.                                                          java.rmi.RemoteException {
17.         if (weatherCriteria != null) {
18.                     System.out.println("city :" + weatherCriteria.getCity());
19.                     System.out.println("state :" + weatherCriteria.getState());
20.                     System.out.println("country :" + weatherCriteria.getCountry());
21.               }
22.               WeatherStatus status = new WeatherStatus();
23.               status.setTemperature(101);
24.               status.setHumidity(12.5);
25.               status.setDescription("Today may be rained or may not be");
26.               return status;
27.       }
28.
29.   }
```

Step 5: Move **deploy.wsdd** and **undeploy.wsdd** which got generated as part of your **src** to **WebContent\WEB-INF** directory



web.xml

```
1. <?xml version="1.0" encoding="ISO-8859-1"?>
2.
3. <!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web
4. Application 2.3//EN" "http://java.sun.com/dtd/web-app_2_3.dtd">
5.
6. <web-app>
7.   <display-name>Apache-Axis</display-name>
8.
9.     <listener>
10.          <listener-
   class>org.apache.axis.transport.http.AxisHTTPSessionListener</listener-class>
11.      </listener>
12.
13.    <servlet>
14.      <servlet-name>AxisServlet</servlet-name>
15.      <display-name>Apache-Axis Servlet</display-name>
16.      <servlet-class>
17.          org.apache.axis.transport.http.AxisServlet
18.      </servlet-class>
19.    </servlet>
20.
21.    <servlet>
22.      <servlet-name>AdminServlet</servlet-name>
23.      <display-name>Axis Admin Servlet</display-name>
```

```
24.       <servlet-class>
25.           org.apache.axis.transport.http.AdminServlet
26.       </servlet-class>
27.       <load-on-startup>100</load-on-startup>
28.     </servlet>
29.
30.     <servlet>
31.       <servlet-name>SOAPMonitorService</servlet-name>
32.       <display-name>SOAPMonitorService</display-name>
33.       <servlet-class>
34.           org.apache.axis.monitor.SOAPMonitorService
35.       </servlet-class>
36.       <init-param>
37.         <param-name>SOAPMonitorPort</param-name>
38.         <param-value>5001</param-value>
39.       </init-param>
40.       <load-on-startup>100</load-on-startup>
41.     </servlet>
42.
43.     <servlet-mapping>
44.       <servlet-name>AxisServlet</servlet-name>
45.       <url-pattern>/servlet/AxisServlet</url-pattern>
46.     </servlet-mapping>
47.
48.     <servlet-mapping>
49.       <servlet-name>AxisServlet</servlet-name>
50.       <url-pattern>*.jws</url-pattern>
51.     </servlet-mapping>
52.
53.     <servlet-mapping>
54.       <servlet-name>AxisServlet</servlet-name>
55.       <url-pattern>/services/*</url-pattern>
56.     </servlet-mapping>
57.
58.     <servlet-mapping>
59.       <servlet-name>SOAPMonitorService</servlet-name>
60.       <url-pattern>/SOAPMonitor</url-pattern>
61.     </servlet-mapping>
62.
63.    <!-- uncomment this if you want the admin servlet -->
64.    <!--
65.     <servlet-mapping>
66.       <servlet-name>AdminServlet</servlet-name>
67.       <url-pattern>/servlet/AdminServlet</url-pattern>
68.     </servlet-mapping>
69.    -->
70.
71.       <session-config>
72.           <!-- Default to 5 minute session timeouts -->
73.           <session-timeout>5</session-timeout>
74.       </session-config>
75.
76.       <!-- currently the W3C havent settled on a media type for WSDL;
77.       http://www.w3.org/TR/2003/WD-wsdl12-20030303/#ietf-draft
78.       for now we go with the basic 'it's XML' response -->
```

```
79.     <mime-mapping>
80.       <extension>wsdl</extension>
81.        <mime-type>text/xml</mime-type>
82.     </mime-mapping>
83.
84.
85.     <mime-mapping>
86.       <extension>xsd</extension>
87.       <mime-type>text/xml</mime-type>
88.     </mime-mapping>
89.
90.     <welcome-file-list id="WelcomeFileList">
91.       <welcome-file>index.jsp</welcome-file>
92.       <welcome-file>index.html</welcome-file>
93.       <welcome-file>index.jws</welcome-file>
94.     </welcome-file-list>
95.
96.   </web-app>
```

4.Deploy our web service into Web server

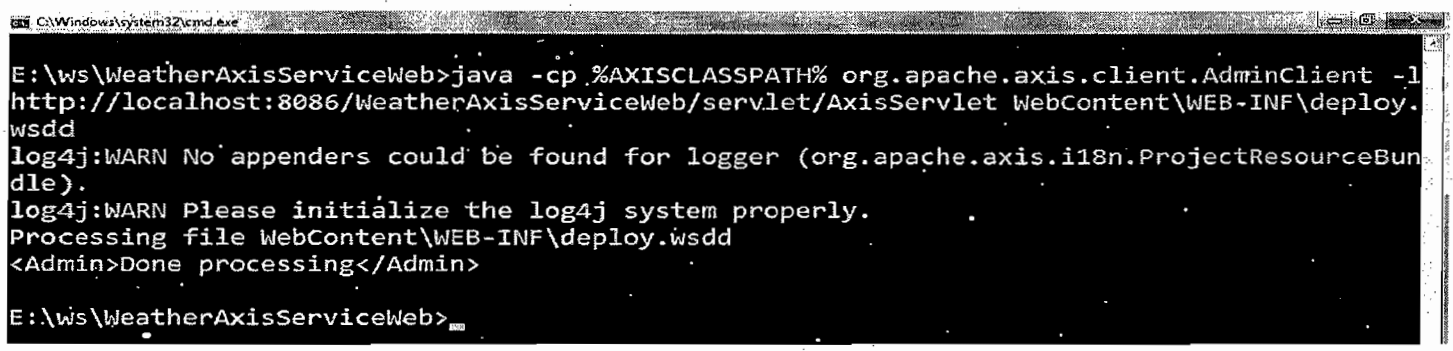
Step 1: With the above changes re-deploy the application again into Web server, And start the server

Step 2: Now type the following command in the command prompt

E:\ws\WeatherAxisServiceWeb>java -cp %AXISCLASSPATH% org.apache.axis.client.AdminClient -lhttp://<remotehost>:<port>/<context-root>/servlet/AxisServlet WebContent\WEB-INF\deploy.wsdd

i.e in our example

E:\ws\WeatherAxisServiceWeb> java -cp %AXISCLASSPATH% org.apache.axis.client.AdminClient -lhttp://**localhost:8086/WeatherAxisServiceWeb**/servlet/AxisServlet WebContent\WEB-INF\deploy.wsdd

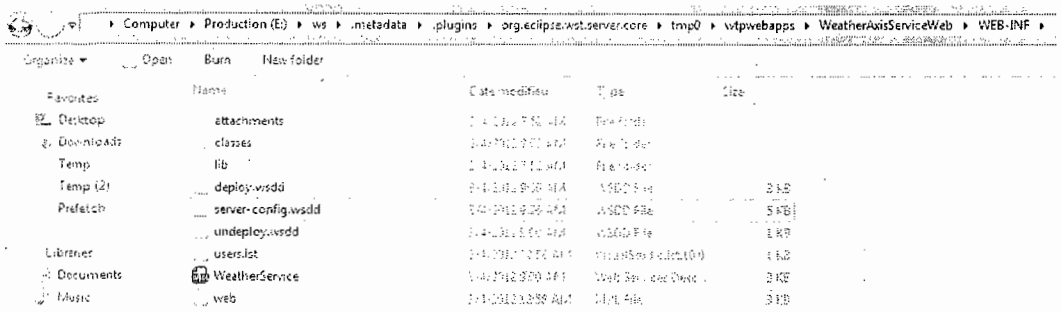


Step3: The above command should pass the **deploy.wsdd** file to **AxisServlet** and should have generated **server-config.wsdd** file

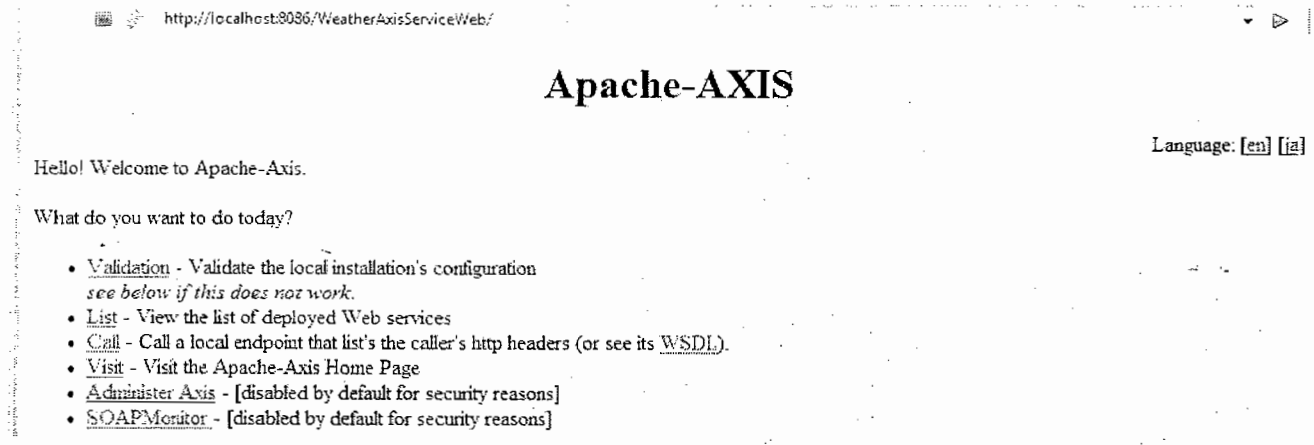
Step 4: Copy the **server-config.wsdd** file which was generated under your **<WORKSPACEDIR>\.metadata\.plugins\org.eclipse.wst.server.core\tmp0\wtpwebapps\<PROJECTCONTEXT-ROOT>\WebContent\WEB-INF** to our project **WEB-INF** directory.

i.e. In our example

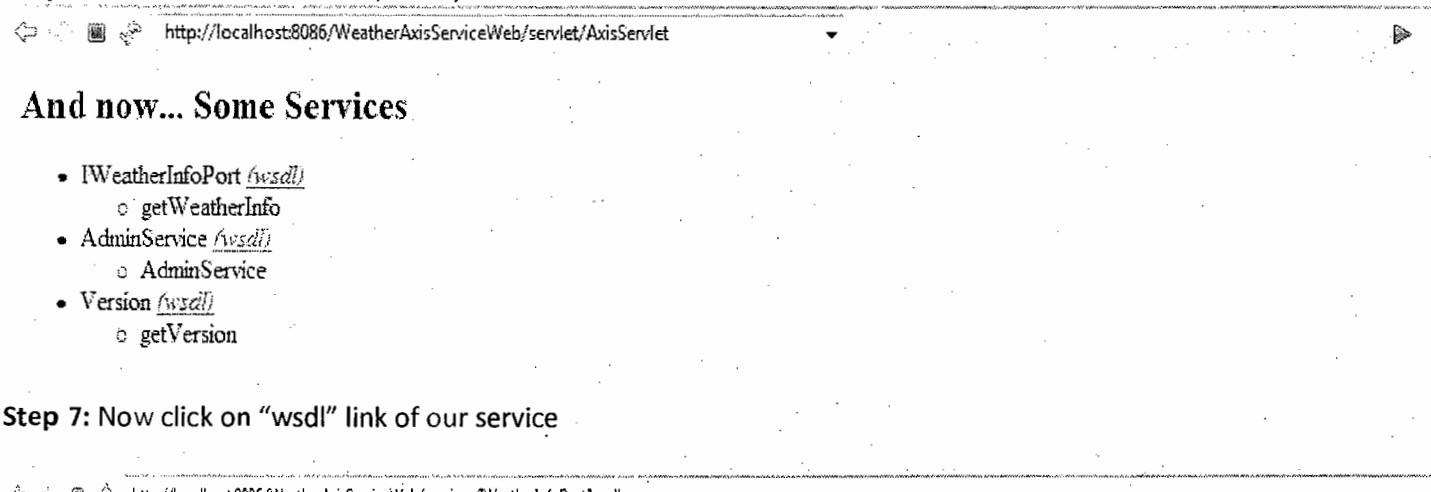
From **E:\ws\.metadata\.plugins\org.eclipse.wst.server.core\tmp0\wtpwebapps\WeatherAxisServiceWeb\WEB-INF\ server-config.wsdd** to our **project WEB-INF** folder



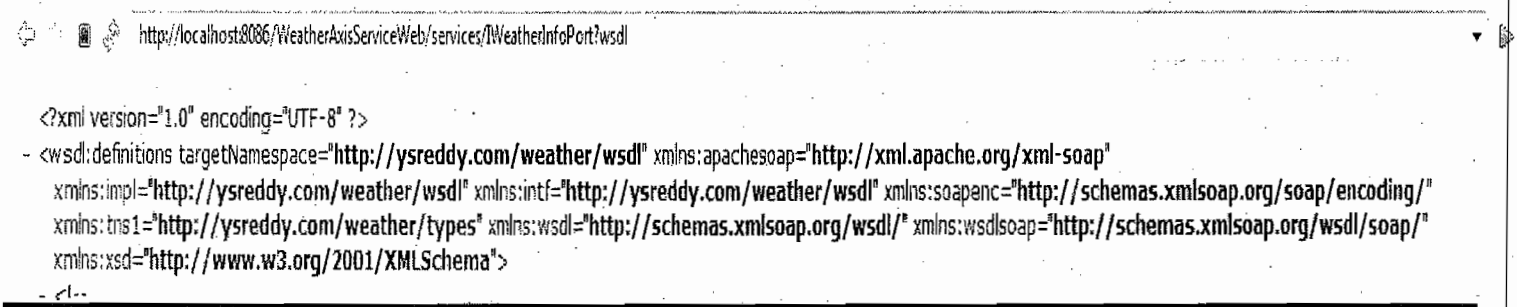
Step 5: Now redeploy our application into web server and access happy axis homepage with the following link **http://localhost:8086/WeatherAxisServiceWeb/**



Step 6: Now click on "**List**" link, now you should see our web service in the list



Step 7: Now click on "wsdl" link of our service



Step 8: Create the Client and call our Service

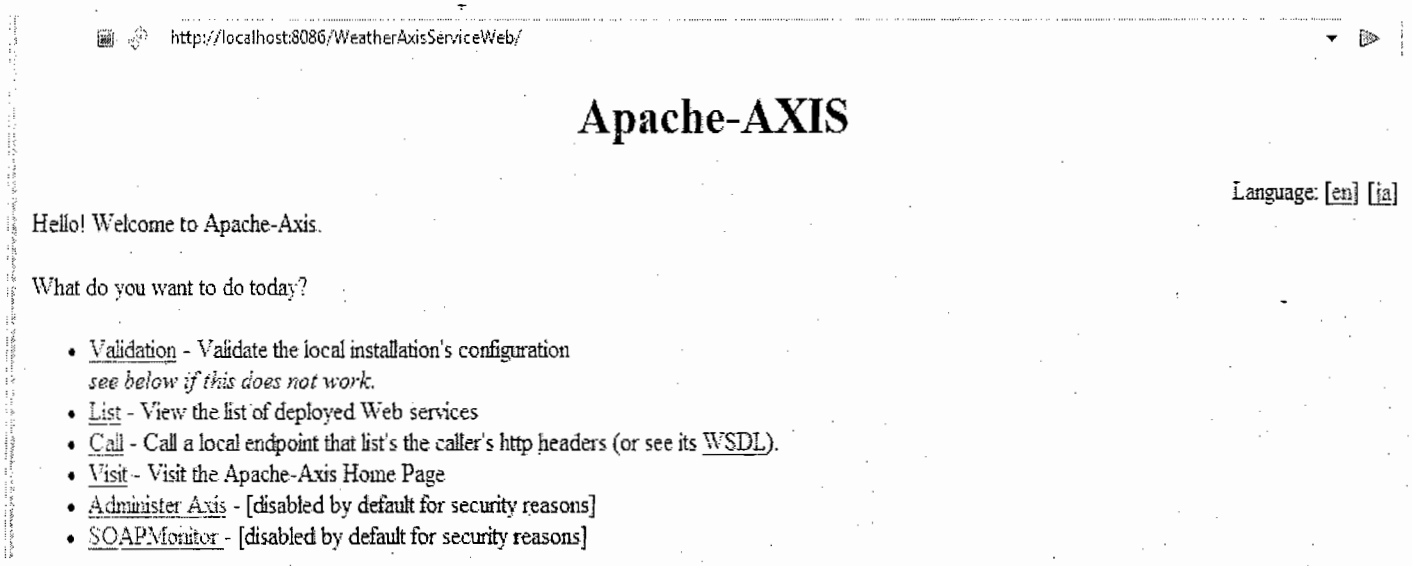
Step 9: If we want we can undeploy the service without stopping the serve as follows.

E:\ws\WeatherAxisServiceWeb>java -cp %AXISCLASSPATH% org.apache.axis.client.AdminClient -lhttp://<remotehost>:<port>/<context-root>/servlet/AxisServlet WebContent\WEB-INF\undeploy.wsdd

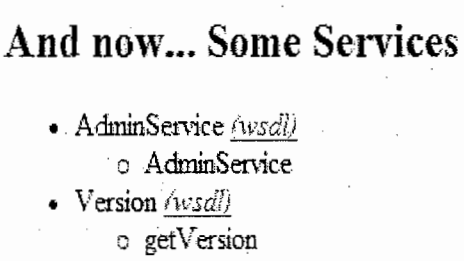
i.e in our example

E:\ws\WeatherAxisServiceWeb> java -cp %AXISCLASSPATH% org.apache.axis.client.AdminClient -lhttp://**localhost:8086/WeatherAxisServiceWeb**/servlet/AxisServlet WebContent\WEB-INF\undeploy.wsdd

Now access happy axis homepage with the following link
http://localhost:8086/WeatherAxisServiceWeb/



Now click on "**List**" link, now you should not see our web service in the list



JAX-WS

- JAX-WS stands for Java API for XML-based web services
- JAX-RPC 2.0 specification is an extension of JAX-RPC 1.1
- Later JAX-RPC 2.0 renamed to JAX-WS 2.0
- But generally we say, the successor to JAX-RPC 1.1 is JAX-WS 2.0

Q.) What are the similarities/differences between JAX-RPC and JAX-WS?

PROPERTY	JAX-RPC	JAX-WS
HTTP protocol Support	HTTP 1.1	HTTP 1.1
SOAP protocol support	SOAP 1.1	Both SOAP 1.1, SOAP 1.2
WSDL version support	WSDL 1.1	Both WSDL 1.1 & WSDL 2.0
Message Format support	SOAP over HTTP	Both SOAP over HTTP & XML over HTTP
WS-I's Basic Profile	BP 1.0	BP 1.1
JDK Version	JDK 1.4	JDK 5.0 and higher (Relies on features like annotations, generics, executors)
J2EE Version	J2EE 1.4(supports JAX-RPC)	J2EE 5(supports both JAX-RPC, JAX-WS)
Binding API (marshall/unmarshall)	Supports any binding API	Supports only JAX-B binding API
MEP(Message Exchanging Pattern)	Supports only synchronous web services	Supports both Synchronous and Asynchronous web services
message-oriented functionality	Doesn't support	Supports
Attachments API	Supports only SAAJ	Supports both SAAJ, MTOM (Message Transmission Optimization Mechanism)
Handlers	Rely on SAAJ 1.2 specification	Rely on SAAJ 1.3 specification

Here are few reasons why we would or would not want to move to JAX-WS form JAX-RPC

Reasons we may want to stay with JAX-RPC 1.1

- If we don't want to step up to Java 5
- If we want to send SOAP encoded messages or create RPC/encoded style WSDL

- Still support is there for JAX-RPC

Reasons to step up to JAX-WS 2.0

- If we want to use messaged-oriented API's
- If we want to use MTOM to send attachments
- If we want better support for XML schema through JAXB
- If we want to use an asynchronous programming model
- If we need to have clients or services that can handle SOAP 1.2 messages
- If we want to eliminate the need for SOAP in web services and just use XML/HTTP binding
- If we would like play with leading edge technologies

Specification (from WS-I)	API (from Sun microsystem)	Implementation (from different companies)
BP 1.1	JAX-WS	JAX-WS RI(from Sun) Apache-Axis2(from ASF) Metro(from Sun) CXF(from ASF)...etc.

- JAX-WS API hides the complexity of web services development
- With the help of annotations runtime environment will take care of creation most of the artifacts
- If we are using JAVA 5, then we need to physically add the jar files JAX-WS API and implementation jar files
- If we are using JAVA 6, then as part of jdk itself we have JAX-WS API and just we need to add jar files of implementation.
- Writing client for JAX-WS also very simple

JAX-WS provide following tools to work with web services for provider and consumer.

1. wsimport
2. wsgen

1.)wsimport

The wsimport tool generates JAX-WS portable artifacts, such as:

- Service Endpoint Interface (SEI)
- Service
- Exception class mapped from wsdl:fault (if any)
- Async Reponse Bean derived from response wsdl:message (if any)
- JAXB generated value types (mapped java classes from schema types)

Command-line : Syntax

wsimport [options] <wsdl>

Option	Description
-d <directory>	Specify where to place generated output files
-b <path>	Specify external JAX-WS or JAXB binding files (Each <file> must have its own -b)
-catalog	Specify catalog file to resolve external entity references, it supports TR9401, XCatalog, and OASIS XML Catalog format. Please read the XML Entity and URI Resolvers document or see wsimport_catalog sample.
-extension	allow vendor extensions (functionality not specified by the specification). Use of extensions may result in applications that are not portable or may not interoperate with other implementations
-help	Display help
-httpproxy:<host>:<port>	Specify an HTTP proxy server (port defaults to 8080)
-keep	Keep generated files
-p	Specifying a target package via this command-line option, overrides any wsdl and schema binding customization for package name and the default package name algorithm defined in the specification
-s <directory>	Specify where to place generated source files
-verbose	Output messages about what the compiler is doing
-version	Print version information
-wsdllocation <location>	@WebService.wsdlLocation and @WebServiceClient.wsdlLocation value

JAX-WS : wsimport tool example

The wsimport tool is used to parse an existing Web Services Description Language (WSDL) file and generate required files (JAX-WS portable artifacts) for web service client to access the published web services. This wsimporttool is available in the $JDK/bin folder.

1.Server – Published web service – WSDL file.

The CompA has published a web service along with a WSDL file at URL
: ***http://compA.com/ws/server?wsdl***

2. Client – Access the published service.

For CompB, to develop a web service client to access the CompA published web service, they can use wsimport tool to parse CompA's WSDL file and generate files (JAX-WS portable artifacts) to access CompA's published service.

Command : wsimport command to parse CompA WSDL file

```
C:\>wsimport -keep -verbose
http://compA.com/ws/server?wsdl
parsing WSDL...

generating code...
com\ysreddy\ws\ServerInfo.java
com\ ysreddy\ws\ServerInfoImplService.java
```

2.)wsgen

The wsgen tool generates JAX-WS portable artifacts used in JAX-WS web services. The tool reads a web service endpoint class and generates all the required artifacts for web service deployment, and invocation. JAX-WS

Command-line : Syntax

```
wsgen [options] <SEI>
```

Option	Description
-classpath <path>	Specify where to find input class files
-cp <path>	Same as -classpath <path>
-d <directory>	Specify where to place generated output files
-extension	allow vendor extensions (functionality not specified by the specification). Use of extensions may result in applications that are not portable or may not interoperate with other implementations
-help	Display help
-keep	Keep generated files
-r <directory>	Used only in conjunction with the -wsdl option. Specify where to place generated

	resource files such as WSDLs
-s <directory>	Specify where to place generated source files
-verbose	Output messages about what the compiler is doing
-version	Print version information. Use of this option will ONLY print version information. Normal processing will not occur.
-wsdl[:protocol]	By default wsgen does not generate a WSDL file. This flag is optional and will cause wsgen to generate a WSDL file and is usually only used so that the developer can look at the WSDL before the endpoint is deploy. The protocol is optional and is used to specify what protocol should be used in the wsdl:binding. Valid protocols include: soap1.1 andXsoap1.2. The default is soap1.1. Xsoap1.2 is not standard and can only be used in conjunction with the -extension option.
-servicename <name>	Used only in conjunction with the -wsdl option. Used to specify a particular wsdl:service name to be generated in the WSDL. Example, -servicename "{http://mynamespace/}MyService"
-portname <name>	Used only in conjunction with the -wsdl option. Used to specify a particular wsdl:port name to be generated in the WSDL. Example, -portname "{http://mynamespace/}MyPort"

JAX-WS : wsgen tool example

The wsgen tool is used to parse an existing web service implementation class and generates required files (JAX-WS portable artifacts) for web service deployment. This wsgen tool is available in $JDK/bin folder.

Use cases

2 common use cases for wsgen tool :

1. Generates JAX-WS portable artifacts (Java files) for web service deployment.
2. Generates WSDL and xsd files, for testing or web service client development.

Let's see a web service implementation class, quite simple, just a method to return a string.

File : ServerInfo.java

```
package com.ysreddy.ws;

import javax.jws.WebMethod;
import javax.jws.WebService;

@WebService
public class ServerInfo{

        @WebMethod
        public String getIpAddress() {

                return "10.10.10.10";
```

```
        }

}
```

1. Generates JAX-WS portable artifacts (Java files)

To generate all the JAX-WS portable artifacts for above web service implementation class (ServerInfo.java), use following command :

Command : wsgen usage

```
D:\>wsgen -verbose -keep -cp . com.ysreddy.ws.ServerInfo

Note:   ap round: 1
[ProcessedMethods Class: com.ysreddy.ws.ServerInfo]
[should process method: getIpAddress hasWebMethods: true ]
[endpointReferencesInterface: false]
[declaring class has WebSevice: true]
[returning: true]
[WrapperGen - method: getIpAddress()]
[method.getDeclaringType(): com.ysreddy.ws.ServerInfo]
[requestWrapper: com.ysreddy.ws.jaxws.GetIpAddress]
[ProcessedMethods Class: java.lang.Object]
com\ysreddy\ws\jaxws\GetIpAddress.java
com\ysreddy\ws\jaxws\GetIpAddressResponse.java
Note:   ap round: 2
```

In this case, it generated four files :

1. com\ysreddy\ws\jaxws\GetIpAddress.java
2. com\ysreddy\ws\jaxws\GetIpAddress.class
3. com\ysreddy\ws\jaxws\GetIpAddressResponse.java
4. com\ysreddy\ws\jaxws\GetIpAddressResponse.class

File : GetIpAddress.java

```
package com.ysreddy.ws.jaxws;

import javax.xml.bind.annotation.XmlAccessType;
import javax.xml.bind.annotation.XmlAccessorType;
import javax.xml.bind.annotation.XmlRootElement;
import javax.xml.bind.annotation.XmlType;

@XmlRootElement(name = "getIpAddress", namespace
= "http://ws.ysreddy.com/")
@XmlAccessorType(XmlAccessType.FIELD)
@XmlType(name = "getIpAddress", namespace =
"http://ws.ysreddy.com/")
public class GetIpAddress {

}
```

File : GetIpAddressResponse.java

```
package com.ysreddy.ws.jaxws;

import javax.xml.bind.annotation.XmlAccessType;
import javax.xml.bind.annotation.XmlAccessorType;
import javax.xml.bind.annotation.XmlElement;
import javax.xml.bind.annotation.XmlRootElement;
import javax.xml.bind.annotation.XmlType;

@XmlRootElement(name = "getIpAddressResponse", namespace =
"http://ws.ysreddy.com/")
@XmlAccessorType(XmlAccessType.FIELD)
@XmlType(name = "getIpAddressResponse", namespace =
"http://ws.ysreddy.com/")
public class GetIpAddressResponse {

    @XmlElement(name = "return", namespace = "")
    private String _return;

    /**
     *
     * @return
     *     returns String
     */
    public String getReturn() {
        return this._return;
    }

    /**
     *
     * @param _return
     *     the value for the _return property
     */
    public void setReturn(String _return) {
        this._return = _return;
    }

}
```

2. Genarates WSDL and xsd

To generate WSDL and xsd files for above web service implementation class (`ServerInfo.java`), add an extra **-wsdl** in the `wsgen` command :

Command : wsgen usage

```
D:\>wsgen -verbose -keep -cp .
com.ysreddy.ws.ServerInfo -wsdl

Note:   ap round: 1
[ProcessedMethods Class:
com.ysreddy.ws.ServerInfo]
[should process method: getIpAddress
hasWebMethods: true ]
[endpointReferencesInterface: false]
[declaring class has WebSevice: true]
[returning: true]
[WrapperGen - method: getIpAddress()]
```

```
[method.getDeclaringType():
com.ysreddy.ws.ServerInfo]
[requestWrapper:
com.ysreddy.ws.jaxws.GetIpAddress]
[ProcessedMethods Class: java.lang.Object]
com\ysreddy\ws\jaxws\GetIpAddress.java
com\ysreddy\ws\jaxws\GetIpAddressResponse.java
Note:   ap round: 2
```

In this case, it generated six files :

1. com\ysreddy\ws\jaxws\GetIpAddress.java
2. com\ysreddy\ws\jaxws\GetIpAddress.class
3. com\ysreddy\ws\jaxws\GetIpAddressResponse.java
4. com\ysreddy\ws\jaxws\GetIpAddressResponse.class
5. ServerInfoService_schema1.xsd
6. ServerInfoService.wsdl

File : ServerInfoService_schema1.xsd

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<xs:schema version="1.0"
targetNamespace="http://ws.ysreddy.com/"
xmlns:tns="http://ws.ysreddy.com/"
xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xs:element name="getIpAddress" type="tns:getIpAddress"/>

  <xs:element name="getIpAddressResponse" type="tns:getIpAddressResponse"/>

  <xs:complexType name="getIpAddress">
    <xs:sequence/>
  </xs:complexType>

  <xs:complexType name="getIpAddressResponse">
    <xs:sequence>
      <xs:element name="return" type="xs:string" minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>
</xs:schema>
```

File : ServerInfoService.wsdl

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<definitions targetNamespace="http://ws.ysreddy.com/"
name="ServerInfoService" xmlns="http://schemas.xmlsoap.org/wsdl/"
xmlns:tns="http://ws.ysreddy.com/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/">
  <types>
    <xsd:schema>
      <xsd:import namespace="http://ws.ysreddy.com/"
                  schemaLocation="ServerInfoService_schema1.xsd"/>
    </xsd:schema>
  </types>
  <message name="getIpAddress">
    <part name="parameters" element="tns:getIpAddress"/>
  </message>
  <message name="getIpAddressResponse">
```

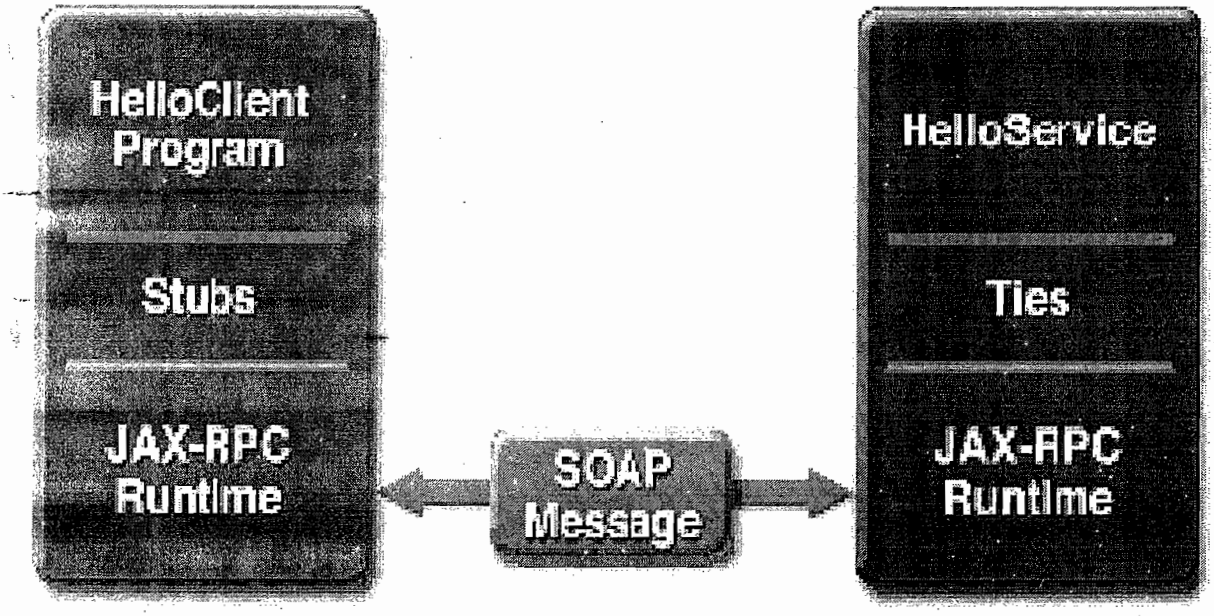
```
    <part name="parameters" element="tns:getIpAddressResponse"/>
  </message>
  <portType name="ServerInfo">
    <operation name="getIpAddress">
      <input message="tns:getIpAddress"/>
      <output message="tns:getIpAddressResponse"/>
    </operation>
  </portType>
  <binding name="ServerInfoPortBinding" type="tns:ServerInfo">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http"
style="document"/>
    <operation name="getIpAddress">
      <soap:operation soapAction=""/>
      <input>
        <soap:body use="literal"/>
      </input>
      <output>
        <soap:body use="literal"/>
      </output>
    </operation>
  </binding>
  <service name="ServerInfoService">
    <port name="ServerInfoPort" binding="tns:ServerInfoPortBinding">
      <soap:address location="REPLACE_WITH_ACTUAL_URL"/>
    </port>
  </service>
</definitions>
```

Requirements of a JAX-WS Endpoint

JAX-WS endpoints must follow these requirements:

- The implementing class must be annotated with either the **javax.jws.WebService** or **javax.jws.WebServiceProvider** annotation.
- The implementing class may explicitly reference an SEI through the **endpointInterface** element of the **@WebService** annotation, but is not required to do so. If no endpointInterface is not specified in **@WebService**, an SEI is implicityly defined for the implementing class.
- The business methods of the implementing class must be public, and must not be declared static or final.
- Business methods that are exposed to web service clients must be annotated with **javax.jws.WebMethod**.
- Business methods that are exposed to web service clients must have JAX-B-compatible parameters and return types. See Default Data Type Bindings.
- The implementing class must not be declared final and must not be abstract.
- The implementing class must have a default public constructor.
- The implementing class must not define the finalize method.
- The implementing class may use **javax.annotation.PostConstruct** or **javax.annotation.PreDestroy** annotations on its methods for lifecycle event callbacks.
- The **@PostConstruct** method is called by the container before the implementing class begins responding to web service clients.
- The **@PreDestroy** method is called by the container before the endpoint is removed from operation.

JAX-WS Architecutre

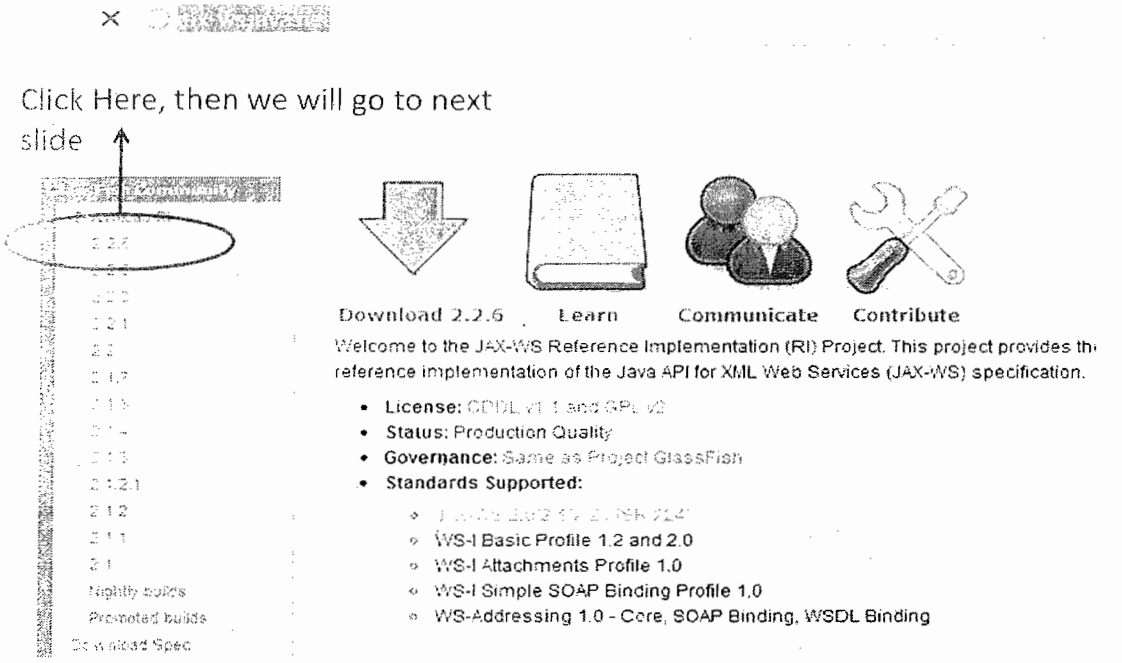


Q.) Develop Web service and client, Using JAX-WS API, Metro implementation?

Step 1: Create a "Dynamic Web Project" with some name like 'RecruitWeb"

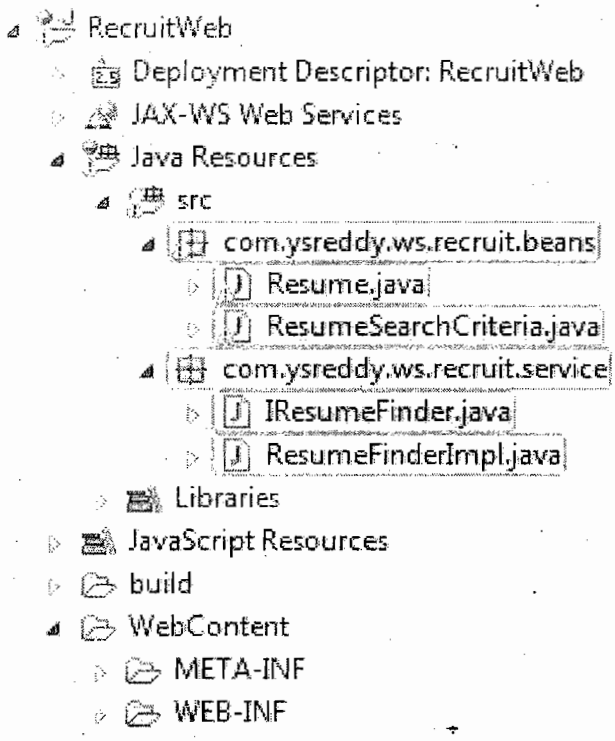
- RecruitWeb
  - Deployment Descriptor: RecruitWeb
  - JAX-WS Web Services
  - Java Resources
    - src
    - Libraries
  - JavaScript Resources
  - build
  - WebContent
    - META-INF
    - WEB-INF
      - lib
      - web.xml

Step 2: download the jar files and add to **WebContent/WEB-INF/lib** folder
Download from : http://jax-ws.java.net/



Now you can find the downloaded file **"JAXWS2.2.6-20120220_1.zip"** file, extract this file then you will find **"jaxws-ri"** folder, inside that you can find **"lib"** folder. In that you can find jar files just copy all the files and paste into our project **"WebContent/WEB-INF/lib"** folder.

Step 3: Develop Input and Output objects, SEI and SEI implementation classes



ResumeSearchCriteria.java

```
1. package com.ysreddy.ws.recruit.beans;
```

```
2.
3. import java.io.Serializable;
4.
5. public class ResumeSearchCriteria implements Serializable {
6.     private int yearsOfExerience;
7.     private String technology;
8.     private double currentCTC;
9.     private double expectedCTC;
10.          private boolean certifed;
11.
12.          // setters & getters
13.
14.    }
```

Resume.java

```
1. package com.ysreddy.ws.recruit.beans;
2.
3. import java.io.Serializable;
4.
5. public class Resume implements Serializable {
6.     private String title;
7.     private String currentCompany;
8.     private String address;
9.     private String description;
10.
11.          // setters & getters
12.
13.    }
```

IResumeFinder.java

```
1. package com.ysreddy.ws.recruit.service;
2.
3. import javax.jws.WebMethod;
4. import javax.jws.WebService;
5. import javax.jws.soap.SOAPBinding;
6. import javax.jws.soap.SOAPBinding.Style;
7.
8. import com.ysreddy.ws.recruit.beans.Resume;
9. import com.ysreddy.ws.recruit.beans.ResumeSearchCriteria;
10.
11.    @WebService
12.    @SOAPBinding(style = Style.RPC)
13.    public interface IResumeFinder {
14.          @WebMethod Resume findBestResume(ResumeSearchCriteria criteria);
15.    }
```

ResumeFinderImpl.java

```
1. package com.ysreddy.ws.recruit.service;
2.
3. import javax.jws.WebService;
4.
5. import com.ysreddy.ws.recruit.beans.Resume;
6. import com.ysreddy.ws.recruit.beans.ResumeSearchCriteria;
7.
8. @WebService(endpointInterface = "com.ysreddy.ws.recruit.service.IResumeFinder")
```

```
9. public class ResumeFinderImpl implements IResumeFinder {
10.         @Override
11.         public Resume findBestResume(ResumeSearchCriteria criteria) {
12.               System.out.println("...Resume Search Parameters...");
13.               System.out.println("Experience : " + criteria.getYearsOfExerience());
14.               System.out.println("Technology : " + criteria.getTechnology());
15.               System.out.println("Current CTC : " + criteria.getCurrentCTC());
16.               System.out.println("Expected CTC : " + criteria.getExpectedCTC());
17.               System.out.println("Is certified : " + criteria.isCertifed());
18.
19.               Resume resume = new Resume();
20.               resume.setTitle("5 years java developer");
21.               resume.setCurrentCompany("Tech-Mhindra");
22.               resume.setAddress("Hi-Tech City");
23.               resume.setDescription("Looking for better opportunity,
24.                                                          to present my best");
25.
26.               return resume;
27.         }
28.   }
```

Step 4: Create a web service deployment descriptor, which is also known as **JAX-WS RI deployment descriptor "sun-jaxws.xml"** , place this under **WebContent\WEB-INF** folder

sun-jaxws.xml

```
1. <?xml version="1.0" encoding="UTF-8"?>
2. <endpoints
3.    xmlns="http://java.sun.com/xml/ns/jax-ws/ri/runtime"
4.    version="2.0">
5.    <endpoint
6.        name="ResumeFinder"
7.        implementation="com.ysreddy.ws.recruit.service.ResumeFinderImpl"
8.        url-pattern="/findResume"/>
9. </endpoints>
```

When user access **/findResume** URL path, it will fire the declared web service, which is **ResumeFinderImpl.java**.

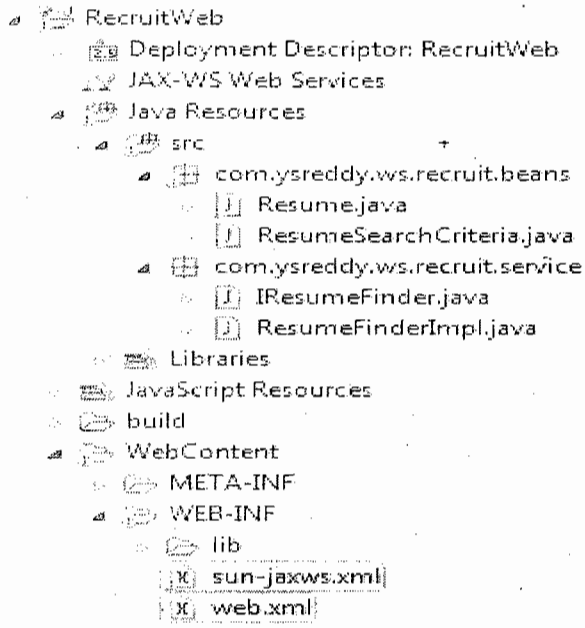
Step 5: open **web.xml** and configure **WSServletContextListener** as listener class, **WSServlet** as our ResumeFinder.

web.xml

```
1. <?xml version="1.0" encoding="UTF-8"?>
2. <web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3. xmlns="http://java.sun.com/xml/ns/javaee"
4. xmlns:web="http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
5. xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
6. http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd" id="WebApp_ID" version="2.5">
7.   <listener>
8.         <listener-class>
9.               com.sun.xml.ws.transport.http.servlet.WSServletContextListener
10.          </listener-class>
11.   </listener>
```

```
12.
13.         <servlet>
14.               <servlet-name>ResumeFinder</servlet-name>
15.               <servlet-class>
16.                     com.sun.xml.ws.transport.http.servlet.WSServlet
17.           </servlet-class>
18.               <load-on-startup>1</load-on-startup>
19.         </servlet>
20.         <servlet-mapping>
21.               <servlet-name>ResumeFinder</servlet-name>
22.               <url-pattern>/findResume</url-pattern>
23.         </servlet-mapping>
24.
25.   </web-app>
```

Final structure of the project is as follows…



Step 6: Now deploy the project into server and hit the service with the following URL

http://localhost:8086/RecruitWeb/findResume?wsdl

Now you should see the WSDL document as follows….

```
<definitions xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd" xmlns:wsp="http://www.w3.org/ns/ws-policy"
 xmlns:wsp1_2="http://schemas.xmlsoap.org/ws/2004/09/policy" xmlns:wsam="http://www.w3.org/2007/05/addressing/metadata"
 xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/" xmlns:tns="http://service.recruit.ws.ysreddy.com/" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns="http://schemas.xmlsoap.org/wsdl/" targetNamespace="http://service.recruit.ws.ysreddy.com/" name="ResumeFinderImplService">
 <types>
  <xsd:schema>
   <xsd:import namespace="http://service.recruit.ws.ysreddy.com/" schemaLocation="http://localhost:8086/RecruitWeb/findResume?xsd=1"/>
  </xsd:schema>
 </types>
 <message name="findBestResume">
  <part name="arg0" type="tns:resumeSearchCriteria"/>
 </message>
 <message name="findBestResumeResponse">
  <part name="return" type="tns:resume"/>
 </message>
 <portType name="IResumeFinder">
  <operation name="findBestResume">
   <input wsam:Action="http://service.recruit.ws.ysreddy.com/IResumeFinder/findBestResumeRequest" message="tns:findBestResume"/>
   <output wsam:Action="http://service.recruit.ws.ysreddy.com/IResumeFinder/findBestResumeResponse" message="tns:findBestResumeResponse"/>
  </operation>
 </portType>
 <binding name="ResumeFinderImplPortBinding" type="tns:IResumeFinder">
  <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="rpc"/>
  <operation name="findBestResume">
   <soap:operation soapAction=""/>
   <input>
    <soap:body use="literal" namespace="http://service.recruit.ws.ysreddy.com/"/>
   </input>
   <output>
    <soap:body use="literal" namespace="http://service.recruit.ws.ysreddy.com/"/>
   </output>
  </operation>
 </binding>
 <service name="ResumeFinderImplService">
  <port name="ResumeFinderImplPort" binding="tns:ResumeFinderImplPortBinding">
   <soap:address location="http://localhost:8086/RecruitWeb/findResume"/>
  </port>
 </service>
</definitions>
```

NOTE : In the above application we are not creating any server side artifacts. When we deploy our application into server, server itself will create the server side artifacts (If server is compatible).

NOTE: Some servers if our web service is having the following SOAP binding style, it won't create server side artifacts.
@SOAPBinding(style = Style.DOCUMENT, use=Use.LITERAL)

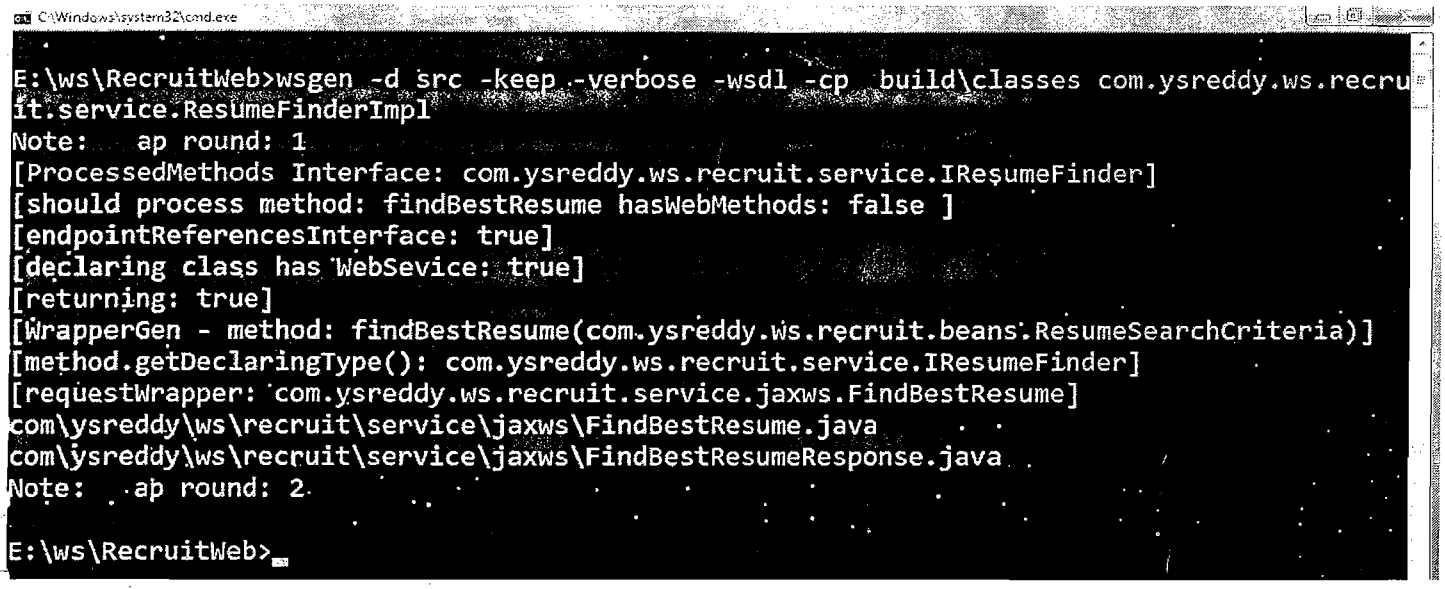
If server unable to create Server side artifacts we will get the following exception, when we are publishing our web service.

```
Exception in thread main com.sun.xml.internal.ws.model.RuntimeModelerException:
        runtime modeler error:

        Wrapper class com.ysreddy.ws.recruit.service.jaxws.FindBestResume is not found.
        Have you run APT to generate them?

        at com.sun.xml.internal.ws.model.RuntimeModeler.getClass(RuntimeModeler.java:156)
        //...
```

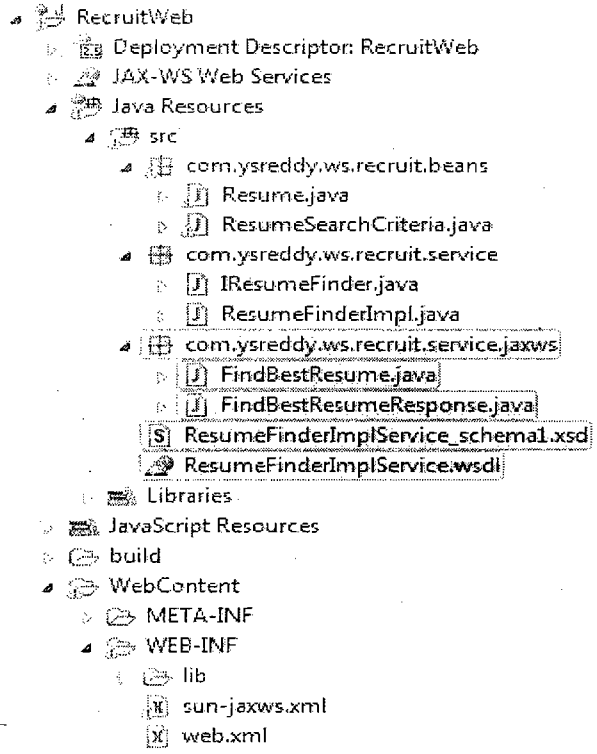
So next steps are required if we want to create server side artifacts on our own.

Step 7: Now open command prompt, and move to up to the root directory of our project, And type the following command to create server side artifacts.

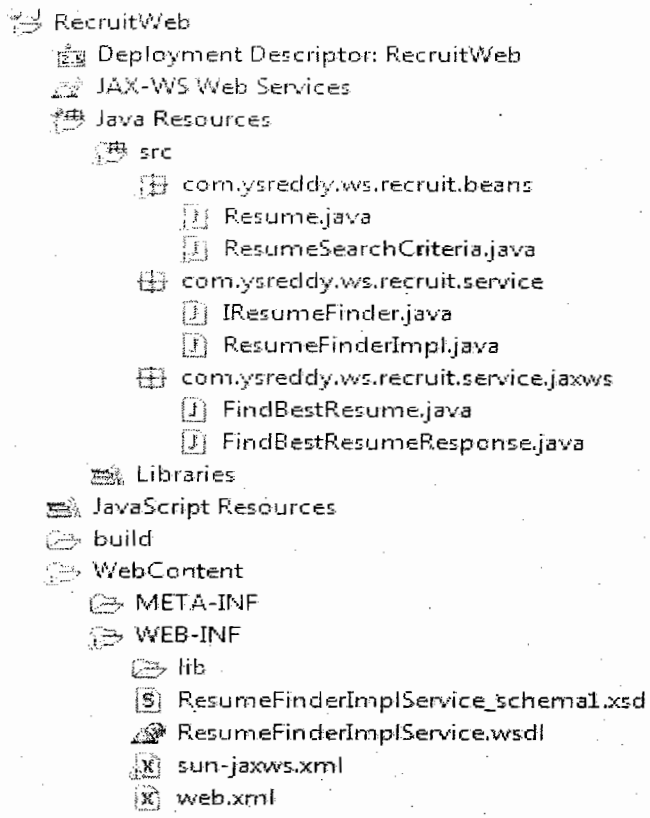
E:\ws\RecruitWeb>**wsgen -d src -keep -verbose -wsdl -cp build\classes com.ysreddy.ws.recruit.service.ResumeFinderImpl**



Step 8: Now go to our project and refresh it you can find the following server side artifacts



tep 9: Now move **WSDL, XSD** files to **WebContent\WEB-INF** folder



Step 10: Now deploy our project into server.

JAX-WS Web Service Clients

Java Web Service Client using wsimport tool

We can use "**wsimport**" tool to parse the published **wsdl** file, and generate necessary client files (stub) to access the published web service.

Where is wsimport?

This wsimport tool is bundle with the JDK, you can find it at "JDK_PATH/bin" folder.

Steps to create JAX-WS Web service Client

Step 1: create a "java project" with any name like "RecruitWebClient", make sure it is pointing to Jdk 6.0

Step 2: open command prompt, and navigate to our project root folder, and make sure that "path" is pointing to jdk 1.6, to test this issue the command follows.

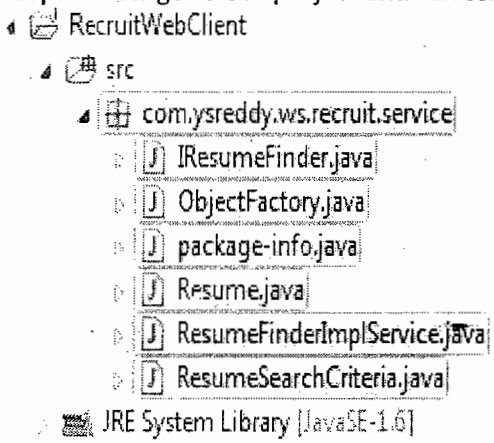
```
C:\Windows\system32\cmd.exe

E:\ws\RecruitWebClient>java -version
java version "1.6.0_18"
Java(TM) SE Runtime Environment (build 1.6.0_18-b07)
Java HotSpot(TM) Client VM (build 16.0-b13, mixed mode, sharing)

E:\ws\RecruitWebClient>
```

Step 3: Now issue the following command, to generate client side artifacts...

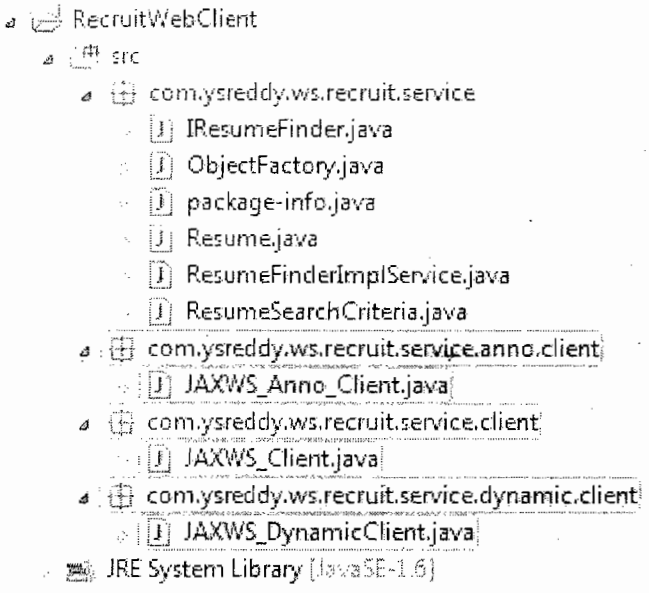
E:\ws\RecruitWebClient> **wsimport -d src -keep -verbose http://localhost:8086/RecruitWeb/findResume?wsdl**

```
E:\ws\RecruitWebClient>wsimport -d src -keep -verbose http://localhost:8086/Recr
dResume?wsdl
parsing WSDL...


generating code...

com\ysreddy\ws\recruit\service\IResumeFinder.java
com\ysreddy\ws\recruit\service\ObjectFactory.java
com\ysreddy\ws\recruit\service\Resume.java
com\ysreddy\ws\recruit\service\ResumeFinderImplService.java
com\ysreddy\ws\recruit\service\ResumeSearchCriteria.java
com\ysreddy\ws\recruit\service\package-info.java

compiling code...
```

Step 4: Now go to our project and refresh the project, we could find client side artifacts as follows



Step 5: Now write the client application



JAXWS Client.java

```
1. package com.ysreddy.ws.recruit.service.client;
2.
3. import com.ysreddy.ws.recruit.service.IResumeFinder;
4. import com.ysreddy.ws.recruit.service.Resume;
5. import com.ysreddy.ws.recruit.service.ResumeFinderImplService;
6. import com.ysreddy.ws.recruit.service.ResumeSearchCriteria;
7.
8. public class JAXWS_Client {
9.     public static void main(String[] args) {
10.             ResumeFinderImplService service = new ResumeFinderImplService();
11.             IResumeFinder resumeFinder = service.getPort(IResumeFinder.class);
12.
13.             ResumeSearchCriteria criteria = new ResumeSearchCriteria();
14.             criteria.setCertifed(true);
15.             criteria.setCurrentCTC(4.5);
16.             criteria.setExpectedCTC(8.5);
17.             criteria.setTechnology("JAVA");
18.             criteria.setYearsOfExerience(5);
19.
20.             Resume resume = resumeFinder.findBestResume(criteria);
21.             System.out.println("....Best Resume matching our requirement....");
22.
23.             System.out.println(resume.getTitle());
24.             System.out.println(resume.getCurrentCompany());
25.             System.out.println(resume.getAddress());
26.             System.out.println(resume.getDescription());
27.     }
28. }
```

Step 6: Now we can write dynamic client(Without client side proxies) as follows...

JAXWS_DynamicClient.java

```
1. package com.ysreddy.ws.recruit.service.dynamic.client;
2.
```

```
3.
4. import java.net.URL;
5.
6. import javax.xml.namespace.QName;
7. import javax.xml.ws.Service;
8.
9. import com.ysreddy.ws.recruit.service.IResumeFinder;
10.   import com.ysreddy.ws.recruit.service.Resume;
11.   import com.ysreddy.ws.recruit.service.ResumeSearchCriteria;
12.
13.   public class JAXWS_DynamicClient {
14.         public static void main(String[] args) throws Exception {
15.
16.               URL url = new URL("http://localhost:8086/RecruitWeb/findResume?wsdl");
17.
18.               // 1st argument service URI, refer to wsdl document above
19.               // 2nd argument is service name, refer to wsdl document above
20.               QName qname = new QName("http://service.recruit.ws.ysreddy.com/",
21.                           "ResumeFinderImplService");
22.
23.               Service service = Service.create(url, qname);
24.
25.               IResumeFinder resumeFinder = service.getPort(IResumeFinder.class);
26.               ResumeSearchCriteria criteria = new ResumeSearchCriteria();
27.               criteria.setCertifed(true);
28.               criteria.setCurrentCTC(4.5);
29.               criteria.setExpectedCTC(8.5);
30.               criteria.setTechnology("JAVA");
31.               criteria.setYearsOfExerience(5);
32.
33.               Resume resume = resumeFinder.findBestResume(criteria);
34.               System.out.println("....Best Resume matching our requirement....");
35.
36.               System.out.println(resume.getTitle());
37.               System.out.println(resume.getCurrentCompany());
38.               System.out.println(resume.getAddress());
39.               System.out.println(resume.getDescription());
40.
41.         }
42.   }
```

JAXWS Anno Client.java

```
1. package com.ysreddy.ws.recruit.service.anno.client;
2.
3. import javax.xml.ws.WebServiceRef;
4.
5. import com.ysreddy.ws.recruit.service.IResumeFinder;
6. import com.ysreddy.ws.recruit.service.Resume;
7. import com.ysreddy.ws.recruit.service.ResumeFinderImplService;
8. import com.ysreddy.ws.recruit.service.ResumeSearchCriteria;
9.
10.   public class JAXWS_Anno_Client {
11.         @WebServiceRef(
12.                     wsdlLocation = "http://localhost:8086/RecruitWeb/findResume?wsdl")
13.         public static ResumeFinderImplService service;
```

```
14.
15.         public static void main(String[] args) {
16.               JAXWS_Anno_Client client = new JAXWS_Anno_Client();
17.               client.callService();
18.
19.         }
20.
21.         private void callService() {
22.               IResumeFinder resumeFinder = service.getResumeFinderImplPort();
23.
24.               ResumeSearchCriteria criteria = new ResumeSearchCriteria();
25.               criteria.setCertifed(true);
26.               criteria.setCurrentCTC(4.5);
27.               criteria.setExpectedCTC(8.5);
28.               criteria.setTechnology("JAVA");
29.               criteria.setYearsOfExerience(5);
30.
31.               Resume resume = resumeFinder.findBestResume(criteria);
32.               System.out.println("....Best Resume matching our requirement....");
33.
34.               System.out.println(resume.getTitle());
35.               System.out.println(resume.getCurrentCompany());
36.               System.out.println(resume.getAddress());
37.               System.out.println(resume.getDescription());
38.
39.         }
40.   }
```

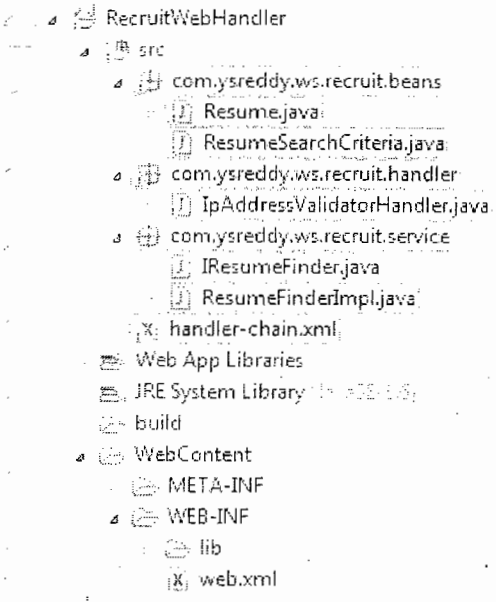
JAX-WS Handler

SOAP handler is a SOAP message interceptor, which is able to intercept incoming or outgoing SOAP message and manipulate its values.

For example, attach a SOAP handler in client side, which will inject client's computer IP address into the SOAP header block for every outgoing SOAP message that is send by the client. In server side, attach another SOAP handler, to retrieve back the client's IP address in SOAP header block from every incoming SOAP message. So that the server side is able to determine which computer is allow to access the published service.

Q.) Develop Webservice application, by applying the SOAP Handler at server side, which will retrieve IP Address from SOAP header block from every incoming SOAP message And do validation to allow only computer with IP address "127.0.0.1" to access this published service. If an invalid client try to access the service, throw a `SOAPFaultException` back to the client.

Step 1: Create a "Dynamic Web Project" with some name like 'RecruitWebHandler"
Step 2: Add required jar files and add to **WebContent/WEB-INF/lib** folder
Step 3: Develop Input and Output objects, SEI , SEI implementation classes, Handler class, Handler configuration file and attach Handler to Service



ResumeSearchCriteria.java

```
1. package com.ysreddy.ws.recruit.beans;
2.
3. import java.io.Serializable;
4.
5. public class ResumeSearchCriteria implements Serializable {
6.     private int yearsOfExerience;
7.     private String technology;
8.     private double currentCTC;
9.     private double expectedCTC;
10.         private boolean certifed;
11.
12.         // setters & getters
13.
14.    }
```

Resume.java

```
1. package com.ysreddy.ws.recruit.beans;
2.
3. import java.io.Serializable;
4.
5. public class Resume implements Serializable {
6.     private String title;
7.     private String currentCompany;
8.     private String address;
9.     private String description;
10.
11.         // setters & getters
12.
13.    }
```

IResumeFinder.java

```
1. package com.ysreddy.ws.recruit.service;
2.
3. import javax.jws.WebMethod;
```

```
4. import javax.jws.WebService;
5. import javax.jws.soap.SOAPBinding;
6. import javax.jws.soap.SOAPBinding.Style;
7.
8. import com.ysreddy.ws.recruit.beans.Resume;
9. import com.ysreddy.ws.recruit.beans.ResumeSearchCriteria;
10.
11.    @WebService
12.    @SOAPBinding(style = Style.RPC)
13.    public interface IResumeFinder {
14.          @WebMethod Resume findBestResume(ResumeSearchCriteria criteria);
15.    }
```

ResumeFinderImpl.java

```
1. package com.ysreddy.ws.recruit.service;
2.
3. import javax.jws.WebService;
4.
5. import com.ysreddy.ws.recruit.beans.Resume;
6. import com.ysreddy.ws.recruit.beans.ResumeSearchCriteria;
7.
8. @WebService(endpointInterface = "com.ysreddy.ws.recruit.service.IResumeFinder")
9. @HandlerChain(file="handler-chain.xml")
10.    public class ResumeFinderImpl implements IResumeFinder {
11.          @Override
12.          public Resume findBestResume(ResumeSearchCriteria criteria) {
13.                System.out.println("...Resume Search Parameters...");
14.                System.out.println("Experience : " + criteria.getYearsOfExerience());
15.                System.out.println("Technology : " + criteria.getTechnology());
16.                System.out.println("Current CTC : " + criteria.getCurrentCTC());
17.                System.out.println("Expected CTC : " + criteria.getExpectedCTC());
18.                System.out.println("Is certified : " + criteria.isCertifed());
19.
20.                Resume resume = new Resume();
21.                resume.setTitle("5 years java developer");
22.                resume.setCurrentCompany("Tech-Mhindra");
23.                resume.setAddress("Hi-Tech City");
24.                resume.setDescription("Looking for better opportunity,
25.                                                      to present my best");
26.
27.                return resume;
28.          }
29.    }
```

IpAddressValidatorHandler.java

```
1. package com.ysreddy.ws.recruit.handler;
2.
3. import java.io.IOException;
4. import java.util.Iterator;
5. import java.util.Set;
6. import javax.xml.namespace.QName;
7. import javax.xml.soap.Node;
8. import javax.xml.soap.SOAPBody;
9. import javax.xml.soap.SOAPConstants;
10.    import javax.xml.soap.SOAPEnvelope;
```

```
import javax.xml.soap.SOAPException;
import javax.xml.soap.SOAPFault;
import javax.xml.soap.SOAPHeader;
import javax.xml.soap.SOAPMessage;
import javax.xml.ws.handler.MessageContext;
import javax.xml.ws.handler.soap.SOAPHandler;
import javax.xml.ws.handler.soap.SOAPMessageContext;
import javax.xml.ws.soap.SOAPFaultException;

public class IpAddressValidatorHandler implements SOAPHandler<SOAPMessageContext>{

      @Override
      public boolean handleMessage(SOAPMessageContext context) {

            System.out.println("Server : handleMessage()......");

Boolean flag = (Boolean) context.get(MessageContext.MESSAGE_OUTBOUND_PROPERTY);

      //for response message only, true for outbound messages, false for inbound
            if(!flag){

                  try{
                        SOAPMessage soapMsg = context.getMessage();
                  SOAPEnvelope soapEnv = soapMsg.getSOAPPart().getEnvelope();
                  SOAPHeader soapHeader = soapEnv.getHeader();

                  //if no header, add one
                  if (soapHeader == null){
                        soapHeader = soapEnv.addHeader();
                        //throw exception
                        generateSOAPErrMessage(soapMsg, "No SOAP header.");
                  }

                  //Get client Ip address from SOAP header
Iterator it = soapHeader.extractHeaderElements(SOAPConstants.URI_SOAP_ACTOR_NEXT);

                  //if no header block for next actor found? throw exception
                  if (it == null || !it.hasNext()){
                generateSOAPErrMessage(soapMsg, "No header block for next actor.");
                  }

                  //if no Ip address found? throw exception
                  Node ipAddressNode = (Node) it.next();
 String ipAddressValue = (ipAddressNode == null) ? null : ipAddressNode.getValue();

                  if (ipAddressValue == null){
            generateSOAPErrMessage(soapMsg, "No Ip address in header block.");
                  }

                  //if Ip address is not match, throw exception
                  if(!ipAddressValue.trim().equals("127.0.0.1")){
      generateSOAPErrMessage(soapMsg, "Invalid Ip address, access is denied.");
                  }

                  //tracking
```

```
66.                     soapMsg.writeTo(System.out);
67.
68.
69.                     }catch(SOAPException e){
70.                          System.err.println(e);
71.                     }catch(IOException e){
72.                          System.err.println(e);
73.                     }
74.
75.               }
76.
77.               //continue other handler chain
78.               return true;
79.          }
80.
81.          @Override
82.          public boolean handleFault(SOAPMessageContext context) {
83.
84.               System.out.println("Server : handleFault()......");
85.
86.               return true;
87.          }
88.
89.          @Override
90.          public void close(MessageContext context) {
91.               System.out.println("Server : close()......");
92.          }
93.
94.          @Override
95.          public Set<QName> getHeaders() {
96.               System.out.println("Server : getHeaders()......");
97.               return null;
98.          }
99.
100.         private void generateSOAPErrMessage(SOAPMessage msg, String reason) {
101.          try {
102.             SOAPBody soapBody = msg.getSOAPPart().getEnvelope().getBody();
103.             SOAPFault soapFault = soapBody.addFault();
104.             soapFault.setFaultString(reason);
105.             throw new SOAPFaultException(soapFault);
106.          }
107.          catch(SOAPException e) { }
108.     }
109.
110. }
```

handler-chain.xml.xml

```
1. <?xml version="1.0" encoding="UTF-8" standalone="yes"?>
2. <javaee:handler-chains xmlns:javaee="http://java.sun.com/xml/ns/javaee"
3.     xmlns:xsd="http://www.w3.org/2001/XMLSchema">
4.     <javaee:handler-chain>
5.       <javaee:handler>
6.          <javaee:handler-class>com.ysreddy.ws.recruit.handler.IpAddressValidatorHandler
7.          </javaee:handler-class>
8.       </javaee:handler>
```

```
    </javaee:handler-chain>
  </javaee:handler-chains>
```

Step 4: Create a web service deployment descriptor, which is also known as **JAX-WS RI deployment descriptor** "**sun-jaxws.xml**" , place this under **WebContent\WEB-INF** folder

sun-jaxws.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<endpoints
   xmlns="http://java.sun.com/xml/ns/jax-ws/ri/runtime"
   version="2.0">
   <endpoint
       name="ResumeFinder"
       implementation="com.ysreddy.ws.recruit.service.ResumeFinderImpl"
       url-pattern="/findResume"/>
</endpoints>
```

When user access **/findResume** URL path, it will fire the declared web service, which is **ResumeFinderImpl.java**.

Step 5: open **web.xml** and configure **WSServletContextListener** as listener class, **WSServlet** as our ResumeFinder.

web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://java.sun.com/xml/ns/javaee"
xmlns:web="http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd" id="WebApp_ID" version="2.5">
  <listener>
        <listener-class>
             com.sun.xml.ws.transport.http.servlet.WSServletContextListener
        </listener-class>
  </listener>

        <servlet>
             <servlet-name>ResumeFinder</servlet-name>
             <servlet-class>
                  com.sun.xml.ws.transport.http.servlet.WSServlet
        </servlet-class>
             <load-on-startup>1</load-on-startup>
        </servlet>
        <servlet-mapping>
             <servlet-name>ResumeFinder</servlet-name>
             <url-pattern>/findResume</url-pattern>
        </servlet-mapping>

  </web-app>
```

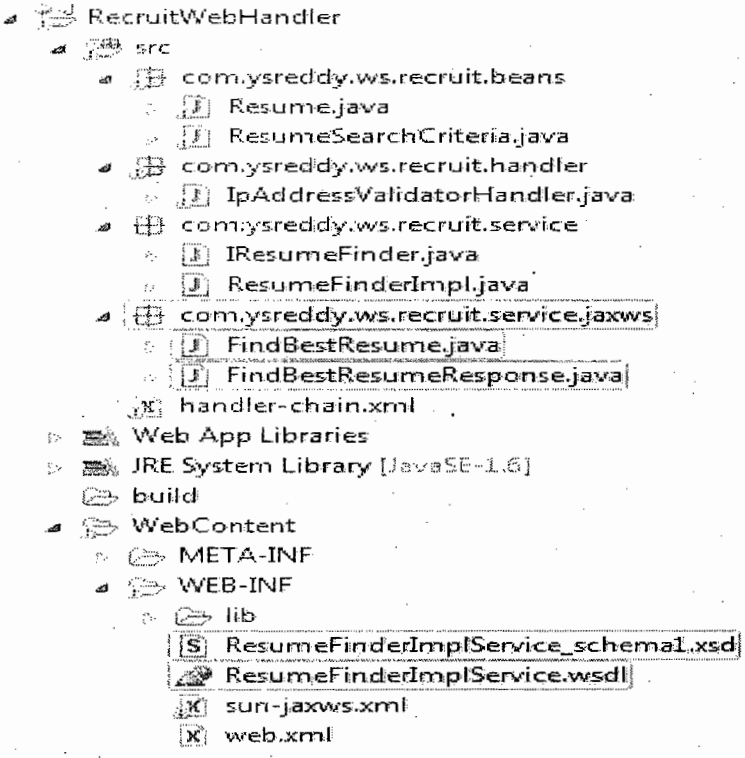
Now the structure of the project is as follows…

Step6: Now open command prompt, and move to up to the root directory of our project, And type the following command to create server side artifacts.

C: \RecruitWebHandler>**wsgen -d src -keep -verbose -wsdl -cp build\classes com.ysreddy.ws.recruit.service.ResumeFinderImpl**

Step 7: Now go to our project and refresh it you can find the following server side artifacts

Step 8: Now move **WSDL, XSD** files to **WebContent\WEB-INF** folder



Step 10: Now deploy our project into server. And hit the service with the following URL http://localhost:8086/RecruitWebHandler/findResume?wsdl Now you should see the WSDL document

Q.) Develop web service client for the above Webservice application, and attach a handler to inject client's IP address into header block, for every outgoing SOAP message that's send by client side.

Step 1: create a "java project" with any name like "RecruitWebClientHandler", make sure it is pointing to Jdk 6.0

Step 2: open command prompt, and navigate to our project root folder, and make sure that "path" is pointing to jdk 1.6.

Step 3: Now issue the following command, to generate client side artifacts...
E:\ RecruitWebClientHandler > **wsimport -d src -keep -verbose http://localhost:8086/RecruitWeb/findResume?wsdl**

Step 4: Now go to our project and refresh the project, we could find client side artifacts

Step 5: Now Develop Client side Handler, Handler configuration file

IpAddressInjectHandler.java

```
1. package com.ysreddy.ws.recruit.service.handler;
2.
3. import java.io.IOException;
4. import java.net.InetAddress;
5. import java.net.UnknownHostException;
6. import java.util.Set;
7.
8. import javax.xml.namespace.QName;
9. import javax.xml.soap.SOAPConstants;
10.   import javax.xml.soap.SOAPEnvelope;
11.   import javax.xml.soap.SOAPException;
12.   import javax.xml.soap.SOAPHeader;
13.   import javax.xml.soap.SOAPHeaderElement;
14.   import javax.xml.soap.SOAPMessage;
15.   import javax.xml.ws.handler.MessageContext;
16.   import javax.xml.ws.handler.soap.SOAPHandler;
17.   import javax.xml.ws.handler.soap.SOAPMessageContext;
18.
19.   public class IpAddressInjectHandler implements SOAPHandler<SOAPMessageContext> {
20.
21.         @Override
22.         public boolean handleMessage(SOAPMessageContext context) {
23.
24.               System.out.println("Client : handleMessage()......");
25.
26.               Boolean isRequest = (Boolean) context
27.                           .get(MessageContext.MESSAGE_OUTBOUND_PROPERTY);
28.
29.               // if this is a request, true for outbound messages, false for inbound
30.               if (isRequest) {
31.
32.                     try {
33.                           SOAPMessage soapMsg = context.getMessage();
34.                           SOAPEnvelope soapEnv = soapMsg.getSOAPPart().getEnvelope();
35.                           SOAPHeader soapHeader = soapEnv.getHeader();
```

```
36.
37.                         // if no header, add one
38.                         if (soapHeader == null) {
39.                               soapHeader = soapEnv.addHeader();
40.                         }
41.
42.                         // get ip address
43.                         String ipAddress = getIpAddress();
44.
45.                         // add a soap header, name as "ip address"
46.                         QName qname = new QName("http://sekharit.com/header/types",
47.                                     "ipAddress");
48.                         SOAPHeaderElement soapHeaderElement =
49.                            soapHeader.addHeaderElement(qname);
50.                   soapHeaderElement.setActor(SOAPConstants.URI_SOAP_ACTOR_NEXT);
51.                         soapHeaderElement.addTextNode(ipAddress);
52.                         soapMsg.saveChanges();
53.
54.                         // tracking
55.                         soapMsg.writeTo(System.out);
56.
57.                   } catch (SOAPException e) {
58.                         System.err.println(e);
59.                   } catch (IOException e) {
60.                         System.err.println(e);
61.                   }
62.
63.             } else {
64.
65.             }
66.
67.             // continue other handler chain
68.             return true;
69.       }
70.
71.       @Override
72.       public boolean handleFault(SOAPMessageContext context) {
73.             System.out.println("Client : handleFault()......");
74.             return true;
75.       }
76.
77.       @Override
78.       public void close(MessageContext context) {
79.             System.out.println("Client : close()......");
80.       }
81.
82.       @Override
83.       public Set<QName> getHeaders() {
84.             System.out.println("Client : getHeaders()......");
85.             return null;
86.       }
87.
88.       // return current client ip address
89.       private String getIpAddress() {
90.             String ipAddress = null;
```

```
91.                 try {
92.                       InetAddress ip = InetAddress.getLocalHost();
93.                       ipAddress = ip.getHostAddress();
94.                       System.out.println("Current IP address : " + ipAddress);
95.                 } catch (UnknownHostException e) {
96.                       e.printStackTrace();
97.                 }
98.
99.                 return ipAddress;
100.          }
101.
102.    }
```

handler-chain.xml

```
1. <?xml version="1.0" encoding="UTF-8" standalone="yes"?>
2. <javaee:handler-chains xmlns:javaee="http://java.sun.com/xml/ns/javaee"
3.     xmlns:xsd="http://www.w3.org/2001/XMLSchema">
4.     <javaee:handler-chain>
5.       <javaee:handler>
6.     <javaee:handler-class>com.ysreddy.ws.recruit.service.handler.IpAddressInjectHandler
7.     </javaee:handler-class>
8.     </javaee:handler>
9.     </javaee:handler-chain>
10.    </javaee:handler-chains>
```

Step 6: Now attach Handler to the client side stub object as follows…

ResumeFinderImplService.java

```
1. @WebServiceClient(name = "ResumeFinderImplService", targetNamespace =
   "http://service.recruit.ws.ysreddy.com/", wsdlLocation =
   "http://localhost:8086/RecruitWebHandler/findResume?wsdl")
2. @HandlerChain(file="handler-chain.xml")
3. public class ResumeFinderImplService
4.     extends Service
5. {
6.     ......
7.     ......
8. }
```

Step 7: Now develop client application

JAXWS Client.java

```
1. package com.ysreddy.ws.recruit.service.client;
2.
3. import com.ysreddy.ws.recruit.service.IResumeFinder;
4. import com.ysreddy.ws.recruit.service.Resume;
5. import com.ysreddy.ws.recruit.service.ResumeFinderImplService;
6. import com.ysreddy.ws.recruit.service.ResumeSearchCriteria;
7.
8. public class JAXWS_Client {
9.     public static void main(String[] args) {
10.              ResumeFinderImplService service = new ResumeFinderImplService();
11.              IResumeFinder resumeFinder = service.getPort(IResumeFinder.class);
12.
```

```
13.             ResumeSearchCriteria criteria = new ResumeSearchCriteria();
14.             criteria.setCertifed(true);
15.             criteria.setCurrentCTC(4.5);
16.             criteria.setExpectedCTC(8.5);
17.             criteria.setTechnology("JAVA");
18.             criteria.setYearsOfExerience(5);
19.
20.             Resume resume = resumeFinder.findBestResume(criteria);
21.             System.out.println("....Best Resume matching our requirement....");
22.
23.             System.out.println(resume.getTitle());
24.             System.out.println(resume.getCurrentCompany());
25.             System.out.println(resume.getAddress());
26.             System.out.println(resume.getDescription());
27.         }
28.  }
```

- RecruitWebClientHandler
  - src
    - com.ysreddy.ws.recruit.service
      - FindBestResume.java
      - FindBestResumeResponse.java
      - IResumeFinder.java
      - ObjectFactory.java
      - package-info.java
      - Resume.java
      - ResumeFinderImplService.java
      - ResumeSearchCriteria.java
    - com.ysreddy.ws.recruit.service.client
      - JAXWS_Client.java
    - com.ysreddy.ws.recruit.service.handler
      - IpAddressInjectHandler.java
    - handler-chain.xml
  - JRE System Library [JavaSE-1.6]

Step 8 : Now run the client application, then it will generate following SOAP request message

```
1. <S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
2.    <S:Header>
3.       <ipAddress xmlns="http://sekharit.com/header/types"
4.       xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
5.       SOAP-ENV:actor="http://schemas.xmlsoap.org/soap/actor/next">
6.          127.0.0.1
7.          </ipAddress>
8.    </S:Header>
9.    <S:Body>
10.             <ns2:findBestResume xmlns:ns2="http://service.recruit.ws.ysreddy.com/">
11.                 <arg0>
12.                     <certifed>true</certifed>
13.                     <currentCTC>4.5</currentCTC>
14.                     <expectedCTC>8.5</expectedCTC>
15.                     <technology>JAVA</technology>
```

```
                        <yearsOfExerience>5</yearsOfExerience>
                    </arg0>
                </ns2:findBestResume>
            </S:Body>
        </S:Envelope>
```

Conclusion

In short, web service handler is just a interceptor to intercept incoming and outgoing SOAP message, both in client or server side, and it's able to manipulate the SOAP message values as well.

Q.) Develop Webservices Applicatioin, where client provides "username" and "password", attached it in SOAP request header and send to server, server parse the SOAP document and retrieve the provided "username" and "password" from request header and do validation ?

NOTE : To run this application we meed to add JAX-WS 2.1 version jars, jdk 6.

Step 1: Create a "Dynamic Web Project" with some name like '**RecruitWebSecured**"
Step 2: Add required jar files and add to **WebContent/WEB-INF/lib** folder
Step 3: Develop Input and Output objects, SEI , SEI implementation classes, SOAP Fault Exception class, sun-jaxws.xml, And configure `WSServletContextListener` as listener class, `WSServlet` as our ResumeFinder.

ResumeSearchCriteria.java

```
1. package com.ysreddy.ws.recruit.beans;
2.
3. import java.io.Serializable;
4.
5. public class ResumeSearchCriteria implements Serializable {
6.     private int yearsOfExerience;
7.     private String technology;
8.     private double currentCTC;
9.     private double expectedCTC;
10.         private boolean certifed;
11.
12.         // setters & getters
13.
14.    }
```

Resume.java

```
1. package com.ysreddy.ws.recruit.beans;
2.
3. import java.io.Serializable;
4.
5. public class Resume implements Serializable {
6.     private String title;
7.     private String currentCompany;
8.     private String address;
9.     private String description;
10.
11.         // setters & getters
```

```
12.
13.   }
```

IResumeFinder.java

```
1. package com.ysreddy.ws.recruit.service;
2.
3. import javax.jws.WebMethod;
4. import javax.jws.WebService;
5. import javax.jws.soap.SOAPBinding;
6. import javax.jws.soap.SOAPBinding.Style;
7. import javax.jws.soap.SOAPBinding.Use;
8.
9. import com.ysreddy.ws.recruit.beans.Resume;
10.   import com.ysreddy.ws.recruit.beans.ResumeSearchCriteria;
11.   import com.ysreddy.ws.recruit.exception.InvalidUserException;
12.
13.   @WebService
14.   @SOAPBinding(style = Style.DOCUMENT, use = Use.LITERAL)
15.   public interface IResumeFinder {
16.         @WebMethod
17.         Resume findBestResume(ResumeSearchCriteria criteria)
18.                     throws InvalidUserException;
19.   }
```

ResumeFinderImpl.java

```
1. package com.ysreddy.ws.recruit.service;
2.
3. import java.util.List;
4. import java.util.Map;
5.
6. import javax.annotation.Resource;
7. import javax.jws.WebService;
8. import javax.xml.ws.WebServiceContext;
9. import javax.xml.ws.handler.MessageContext;
10.
11.   import com.ysreddy.ws.recruit.beans.Resume;
12.   import com.ysreddy.ws.recruit.beans.ResumeSearchCriteria;
13.   import com.ysreddy.ws.recruit.exception.InvalidUserException;
14.
15.   @WebService(endpointInterface = "com.ysreddy.ws.recruit.service.IResumeFinder")
16.   public class ResumeFinderImpl implements IResumeFinder {
17.         @Resource
18.         WebServiceContext wsctx;
19.
20.         @Override
21.         public Resume findBestResume(ResumeSearchCriteria criteria)
22.                     throws InvalidUserException {
23.               System.out.println("...Resume Search Parameters...");
24.               System.out.println("Experience : " + criteria.getYearsOfExerience());
25.               System.out.println("Technology : " + criteria.getTechnology());
26.               System.out.println("Current CTC : " + criteria.getCurrentCTC());
27.               System.out.println("Expected CTC : " + criteria.getExpectedCTC());
28.               System.out.println("Is certified : " + criteria.isCertifed());
29.
30.               validateUser();
```

```
31.
32.             Resume resume = new Resume();
33.             resume.setTitle("5 years java developer");
34.             resume.setCurrentCompany("Tech-Mhindra");
35.             resume.setAddress("Hi-Tech City");
36.       resume.setDescription("Looking for better opportunity, to present my best");
37.
38.             return resume;
39.       }
40.
41.       private void validateUser() throws InvalidUserException {
42.             MessageContext mctx = wsctx.getMessageContext();
43.             // get detail from request headers
44.             Map http_headers = (Map) mctx.get(MessageContext.HTTP_REQUEST_HEADERS);
45.             List userList = (List) http_headers.get("Username");
46.             List passList = (List) http_headers.get("Password");
47.
48.             String username = "";
49.             String password = "";
50.
51.             if (userList != null) {
52.                   // get username
53.                   username = userList.get(0).toString();
54.             }
55.
56.             if (passList != null) {
57.                   // get password
58.                   password = passList.get(0).toString();
59.             }
60.
61.             System.out.println(username);
62.             System.out.println(password);
63.             // Should validate username and password with database
64.             if (!username.equals("sekhar") || !password.equals("sekhar123")) {
65.                   throw new InvalidUserException("Invalid credentials ");
66.             }
67.       }
68. }
```

InvalidUserException.java

```
1. package com.ysreddy.ws.recruit.exception;
2.
3. import javax.xml.ws.WebFault;
4.
5. @WebFault
6. public class InvalidUserException extends Exception{
7.     public InvalidUserException() {
8.     }
9.
10.       public InvalidUserException(String message, Throwable cause) {
11.             super(message, cause);
12.       }
13.
14.       public InvalidUserException(String message) {
15.             super(message);
```

```
16.         }
17.
18.         public InvalidUserException(Throwable cause) {
19.               super(cause);
20.         }
21.
22.   }
```

sun-jaxws.xml

```
1. <?xml version="1.0" encoding="UTF-8"?>
2. <endpoints
3.    xmlns="http://java.sun.com/xml/ns/jax-ws/ri/runtime"
4.    version="2.0">
5.    <endpoint
6.        name="ResumeFinder"
7.        implementation="com.ysreddy.ws.recruit.service.ResumeFinderImpl"
8.        url-pattern="/findResume"/>
9. </endpoints>
```

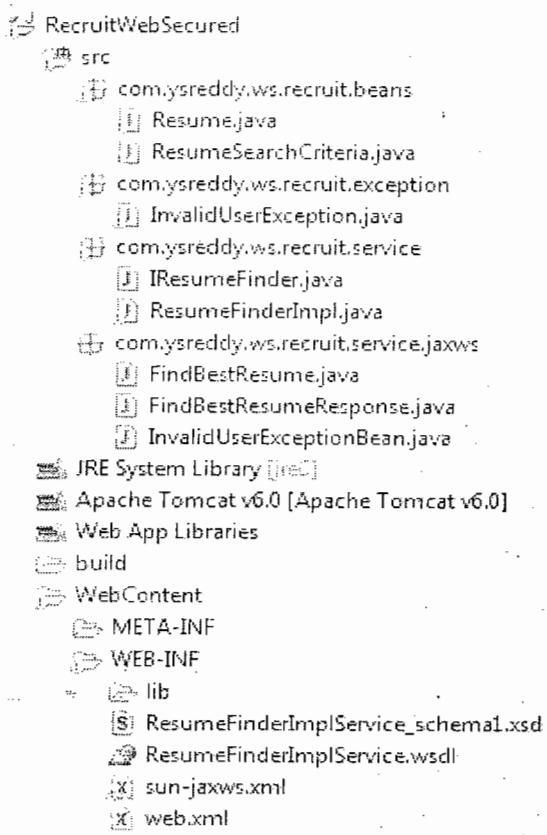
web.xml

```
1. <?xml version="1.0" encoding="UTF-8"?>
2. <web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3. xmlns="http://java.sun.com/xml/ns/javaee"
4. xmlns:web="http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
5. xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
6. http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd" id="WebApp_ID" version="2.5">
7.   <listener>
8.         <listener-class>
9.               com.sun.xml.ws.transport.http.servlet.WSServletContextListener
10.          </listener-class>
11.   </listener>
12.
13.         <servlet>
14.               <servlet-name>ResumeFinder</servlet-name>
15.               <servlet-class>
16.                     com.sun.xml.ws.transport.http.servlet.WSServlet
17.          </servlet-class>
18.               <load-on-startup>1</load-on-startup>
19.         </servlet>
20.         <servlet-mapping>
21.               <servlet-name>ResumeFinder</servlet-name>
22.               <url-pattern>/findResume</url-pattern>
23.         </servlet-mapping>
24.
25.   </web-app>
```

Step 4: Now open command prompt, and move to up to the root directory of our project, And type the following command to create server side artifacts.

E: \RecruitWebSecured>**wsgen -d src -keep -verbose -wsdl -cp build\classes com.ysreddy.ws.recruit.service.ResumeFinderImpl**

Step 5: Now go to our project and refresh it you server side artifacts, Now move **WSDL, XSD** files to **WebContent\WEB-INF** folder, Now the structure of the project is as follows...



Step 6: Now deploy our project into server. And hit the service with the following URL http://localhost:8086/RecruitWebSecured/findResume?wsdl Now you should see the WSDL document

Q.) Develop Webservices client Applicatioin, where client provides "username" and "password", attached it in SOAP request header and send to server?

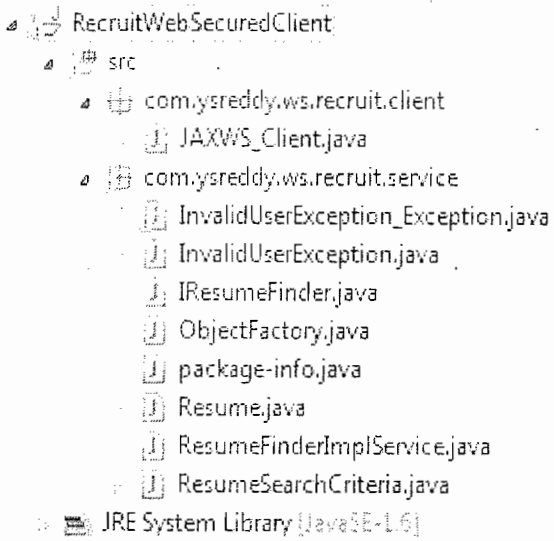
Step 1: create a "java project" with any name like "**RecruitWebSecuredClient**", make sure it is pointing to Jdk 6.0

Step 2: open command prompt, and navigate to our project root folder, and make sure that "path" is pointing to jdk 1.6.

Step 3: Now issue the following command, to generate client side artifacts...
E:\ RecruitWebSecuredClient> **wsimport -d src -keep -verbose http://localhost:8086/RecruitWeb/findResume?wsdl**

Step 4: Now go to our project and refresh the project, we could find client side artifacts

Step 5: Now develop client application

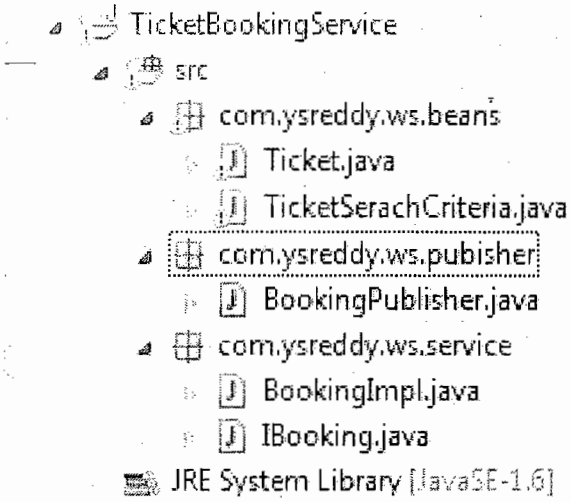


JAXWS_Client.java

```
1. package com.ysreddy.ws.recruit.client;
2.
3. import java.util.Collections;
4. import java.util.HashMap;
5. import java.util.List;
6. import java.util.Map;
7.
8. import javax.xml.ws.BindingProvider;
9. import javax.xml.ws.handler.MessageContext;
10.
11.   import com.ysreddy.ws.recruit.service.IResumeFinder;
12.   import com.ysreddy.ws.recruit.service.InvalidUserException_Exception;
13.   import com.ysreddy.ws.recruit.service.Resume;
14.   import com.ysreddy.ws.recruit.service.ResumeFinderImplService;
15.   import com.ysreddy.ws.recruit.service.ResumeSearchCriteria;
16.
17.   public class JAXWS_Client {
18.         public static void main(String[] args) {
19.               ResumeFinderImplService service = new ResumeFinderImplService();
20.               IResumeFinder resumeFinder = service.getPort(IResumeFinder.class);
21.
22.               ResumeSearchCriteria criteria = new ResumeSearchCriteria();
23.               criteria.setCertifed(true);
24.               criteria.setCurrentCTC(4.5);
25.               criteria.setExpectedCTC(8.5);
26.               criteria.setTechnology("JAVA");
27.               criteria.setYearsOfExerience(5);
28.
29.
30.           /*******************UserName & Password ******************************/
31.    Map<String, Object> req_ctx = ((BindingProvider)resumeFinder).getRequestContext();
32.    req_ctx.put(BindingProvider.ENDPOINT_ADDRESS_PROPERTY,
33.                                      service.getWSDLDocumentLocation().toString());
34.
35.          Map<String, List<String>> headers = new HashMap<String, List<String>>();
```

```
36.           headers.put("Username", Collections.singletonList("sekhar"));
37.           headers.put("Password", Collections.singletonList("sekhar123"));
38.           req_ctx.put(MessageContext.HTTP_REQUEST_HEADERS, headers);
39.           /**************************************************************/
40.
41.               try {
42.                     Resume resume = resumeFinder.findBestResume(criteria);
43.               System.out.println("....Best Resume matching our requirement....");
44.
45.                     System.out.println(resume.getTitle());
46.                     System.out.println(resume.getCurrentCompany());
47.                     System.out.println(resume.getAddress());
48.                     System.out.println(resume.getDescription());
49.               } catch (InvalidUserException_Exception e) {
50.                     System.out.println("Unable to access Webservice. Error Message: "
51.                                                                     +e.getMessage());
52.               }
53.
54.         }
55.   }
56.
```

Q.) Develop a simple stand alone Web service application? Where use JDK given web server to deploy developed webservice?



TicketSerachCriteria.java

```
1. package com.ysreddy.ws.beans;
2.
3. import java.io.Serializable;
4.
5. public class TicketSerachCriteria implements Serializable {
6.     private String date;
7.     private String from;
8.     private String to;
9.
10.         // setters & getters
11.
12.   }
```

Ticket.java

```
1. package com.ysreddy.ws.beans;
2.
3. import java.io.Serializable;
4.
5. public class Ticket implements Serializable {
6.     private int ticketNo;
7.     private double fair;
8.     private String service;
9.
10.         public int getTicketNo() {
11.               return ticketNo;
12.         }
13.
14.         public void setTicketNo(int ticketNo) {
15.               this.ticketNo = ticketNo;
16.         }
17.
18.         public double getFair() {
19.               return fair;
20.         }
21.
22.         public void setFair(double fair) {
23.               this.fair = fair;
24.         }
25.
26.         public String getService() {
27.               return service;
28.         }
29.
30.         public void setService(String service) {
31.               this.service = service;
32.         }
33.
34.   }
```

IBooking.java

```
1. package com.ysreddy.ws.service;
2.
3. import javax.jws.WebMethod;
4. import javax.jws.WebService;
5. import javax.jws.soap.SOAPBinding;
6. import javax.jws.soap.SOAPBinding.Style;
7.
8. import com.ysreddy.ws.beans.Ticket;
9. import com.ysreddy.ws.beans.TicketSerachCriteria;
10.
11.   @WebService
12.   @SOAPBinding(style = Style.RPC)
13.   public interface IBooking {
14.         @WebMethod
15.         public Ticket findTicket(TicketSerachCriteria criteria);
16.
17.   }
```

BookingImpl.java

```
1. package com.ysreddy.ws.service;
2.
3. import javax.jws.WebService;
4.
5. import com.ysreddy.ws.beans.Ticket;
6. import com.ysreddy.ws.beans.TicketSerachCriteria;
7.
8. @WebService(endpointInterface = "com.ysreddy.ws.service.IBooking")
9. public class BookingImpl implements IBooking {
10.         @Override
11.         public Ticket findTicket(TicketSerachCriteria criteria) {
12.               System.out.println("....Search criteria....");
13.               System.out.println("From : " + criteria.getFrom());
14.               System.out.println("To : " + criteria.getTo());
15.               System.out.println("Date : " + criteria.getDate());
16.
17.               Ticket ticket = new Ticket();
18.               ticket.setTicketNo(12345);
19.               ticket.setService("KESINENI");
20.               ticket.setFair(650.89);
21.
22.               return ticket;
23.         }
24.   }
```

BookingPublisher.java

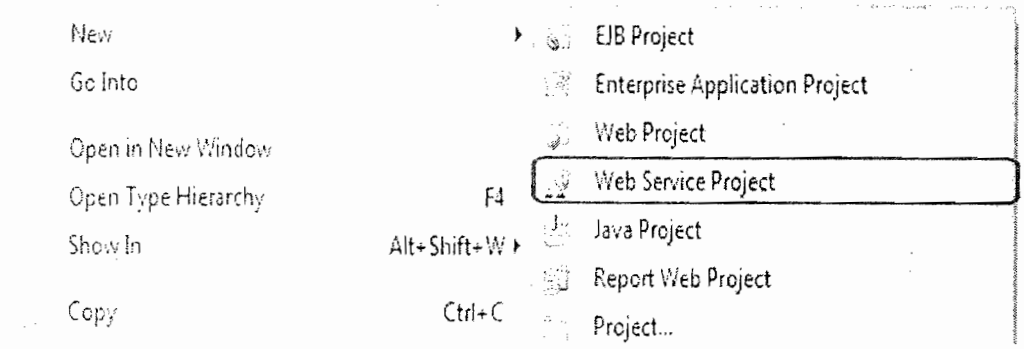
```
1. package com.ysreddy.ws.pubisher;
2.
3. import javax.xml.ws.Endpoint;
4.
5. import com.ysreddy.ws.service.BookingImpl;
6.
7. public class BookingPublisher {
8.    public static void main(String[] args) {
9.     Endpoint.publish("http://localhost:9999/ws/booking/getTicket", new BookingImpl());
10.              System.out.println("....Published....");
11.         }
12.   }
```

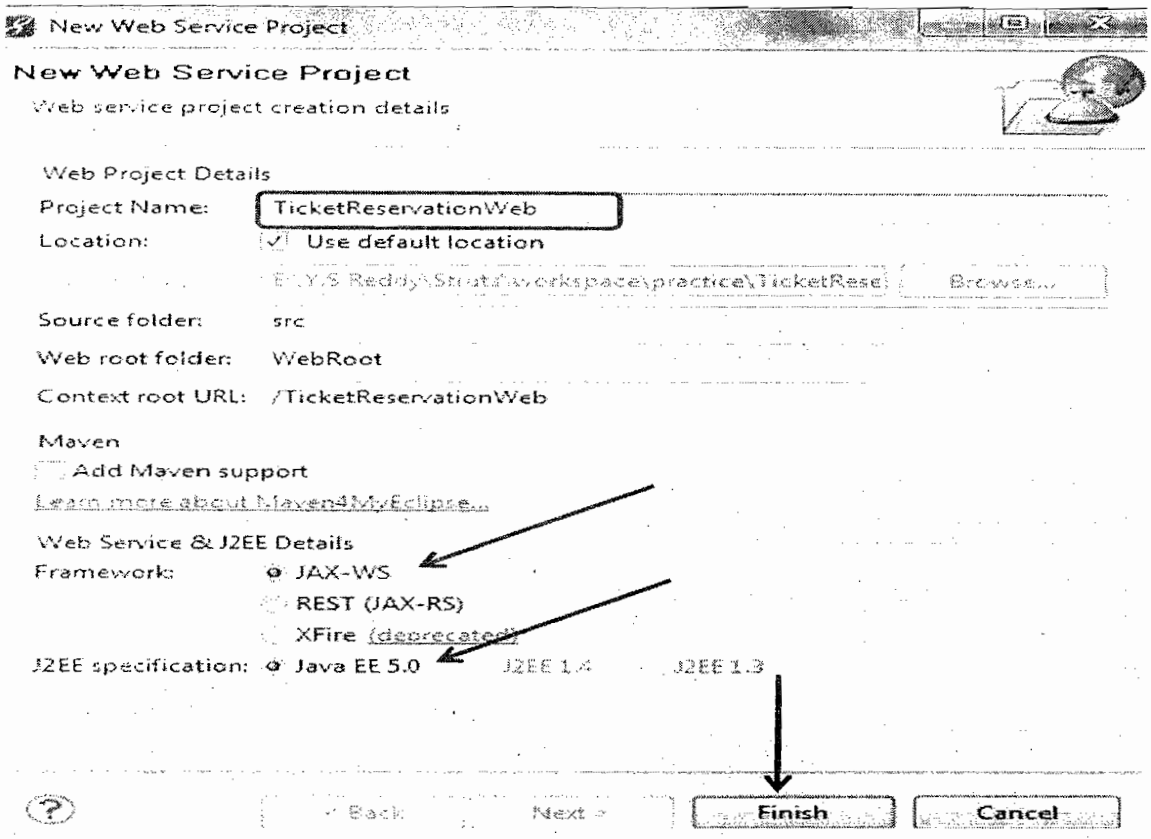
NOTE: Now Run the above Publisher program, then our web service will be deployed into JDK given server, with the given url. And server it self creates required sever side artifacts. Now we can access WSDL with the following URL.

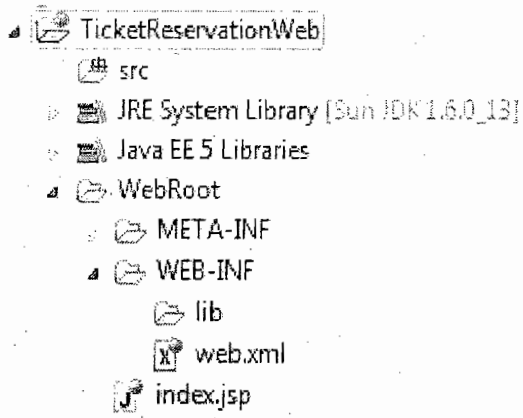
http://localhost:9999/ws/booking/getTicket?wsdl

2.) Develop Web service using Myeclipse(Uses JAX-WS API, Metro implementation)

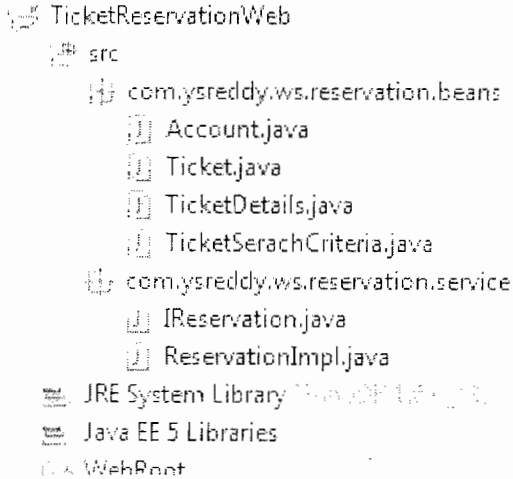
Step 1: Create a "**Web Service Project**" with any like "**TicketReservationWeb**"







Step 2: Now create input, output objects, SEI, SEI implementation classes



Ticket.java

```
1. package com.ysreddy.ws.reservation.beans;
2.
3. import java.io.Serializable;
4.
5. public class Ticket implements Serializable {
6.     private int ticketNo;
7.     private double fair;
8.     private String service;
9.
10.         //setters & getters
11.
12.    }
```

TicketSerachCriteria.java

```
1. package com.ysreddy.ws.reservation.beans;
2.
3. import java.io.Serializable;
4.
5. public class TicketSerachCriteria implements Serializable {
6.     private String date;
7.     private String from;
8.     private String to;
9.
10.         //setters & getters
11.
12.    }
```

Account.java

```
1. package com.ysreddy.ws.reservation.beans;
2.
3. import java.io.Serializable;
4.
5. public class Account implements Serializable {
6.     private int accno;
7.     private String name;
8.
9.     //setters & getters
10.
11.    }
```

TicketDetails.java

```
1. package com.ysreddy.ws.reservation.beans;
2.
3. import java.io.Serializable;
4.
5. public class TicketDetails implements Serializable {
6.     private int seatNo;
7.     private String boardingPoint;
8.     private String time;
9.     private String date;
10.
11.         //setters & getters
12.
13.    }
```

IReservation.java

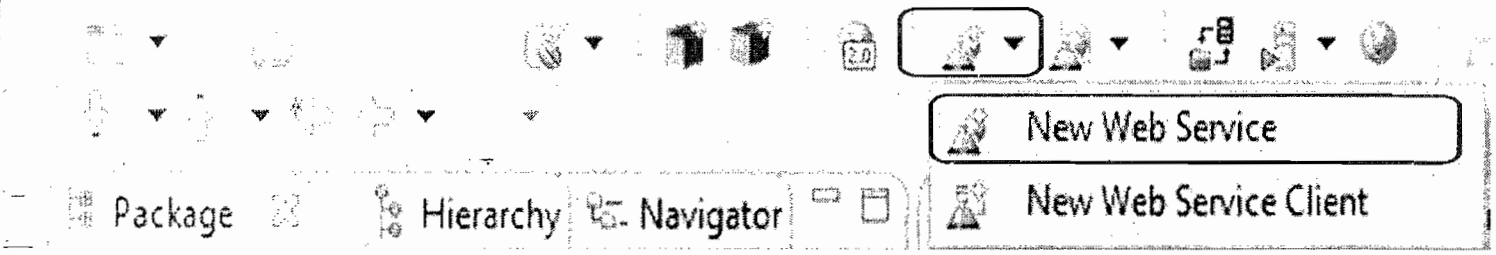
```
1. package com.ysreddy.ws.reservation.service;
2.
3. import com.ysreddy.ws.reservation.beans.Account;
4. import com.ysreddy.ws.reservation.beans.Ticket;
5. import com.ysreddy.ws.reservation.beans.TicketDetails;
6. import com.ysreddy.ws.reservation.beans.TicketSerachCriteria;
7.
8. public interface IReservation {
9.     public Ticket searchTicket(TicketSerachCriteria criteria);
10.
11.         public TicketDetails buyTicket(Account account);
12.    }
```

ReservationImpl.java

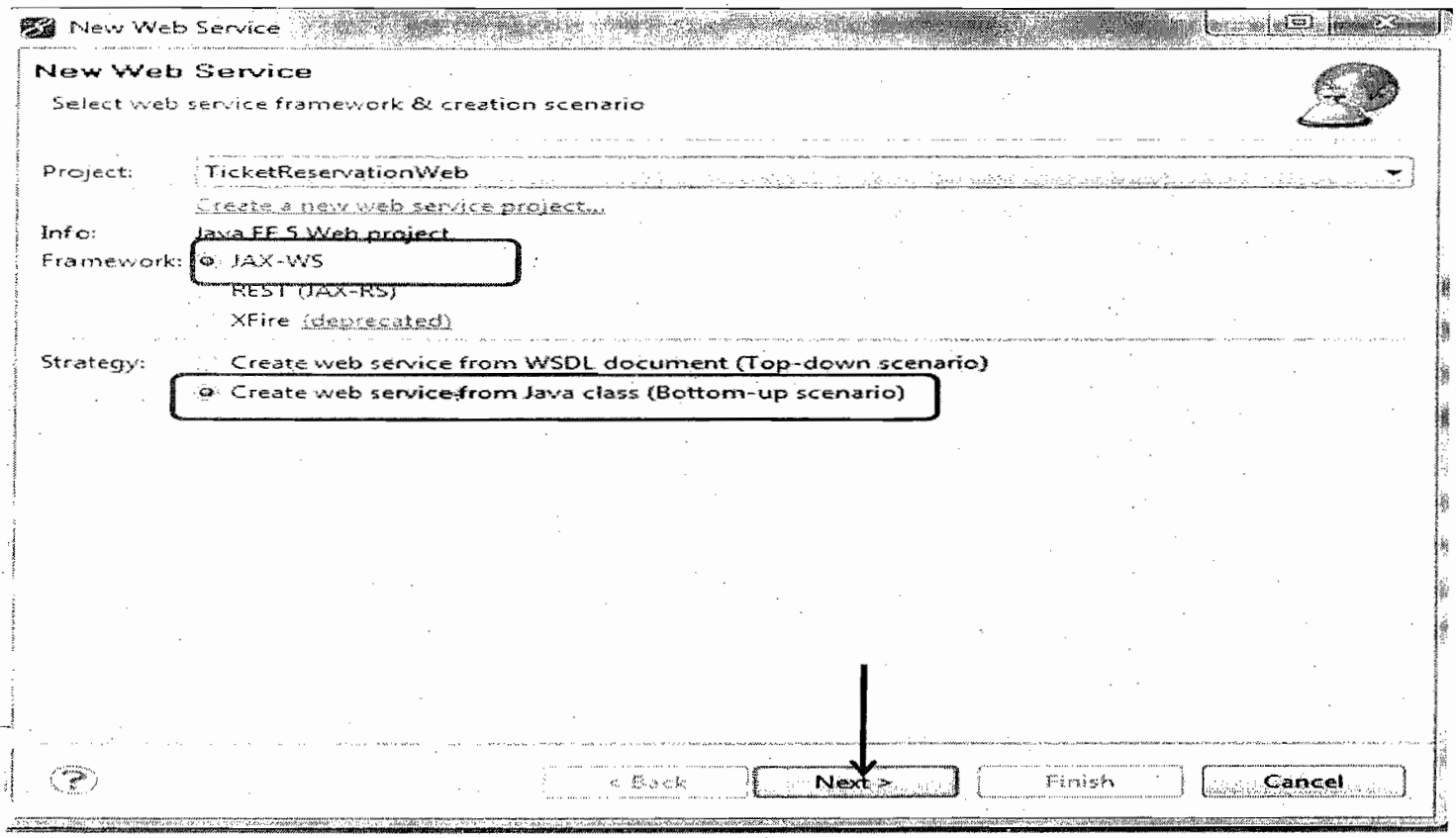
```
1. package com.ysreddy.ws.reservation.service;
2.
3. import com.ysreddy.ws.reservation.beans.Account;
4. import com.ysreddy.ws.reservation.beans.Ticket;
5. import com.ysreddy.ws.reservation.beans.TicketDetails;
6. import com.ysreddy.ws.reservation.beans.TicketSerachCriteria;
7.
8. public class ReservationImpl implements IReservation {
9.     public Ticket searchTicket(TicketSerachCriteria criteria) {
10.                System.out.println("...Search properties...");
11.                System.out.println("TO : " + criteria.getTo());
12.                System.out.println("FROM : " + criteria.getFrom());
13.                System.out.println("DATE : " + criteria.getDate());
14.
15.                Ticket ticket = new Ticket();
16.                ticket.setTicketNo(12345);
17.                ticket.setService("KESINENI");
18.                ticket.setFair(650.98);
19.
20.                return ticket;
21.         }
22.
23.         public TicketDetails buyTicket(Account account) {
```

```
24.                 System.out.println("...Deducting amount from the follwoing account..");
25.                 System.out.println("ACCNO : " + account.getAccno());
26.                 System.out.println("NAME : " + account.getName());
27.
28.                 TicketDetails details = new TicketDetails();
29.                 details.setSeatNo(12345);
30.                 details.setDate("12\03\2012");
31.                 details.setTime("10:45 PM");
32.                 details.setBoardingPoint("S.R.Nagar");
33.
34.                 return details;
35.         }
36.     }
```

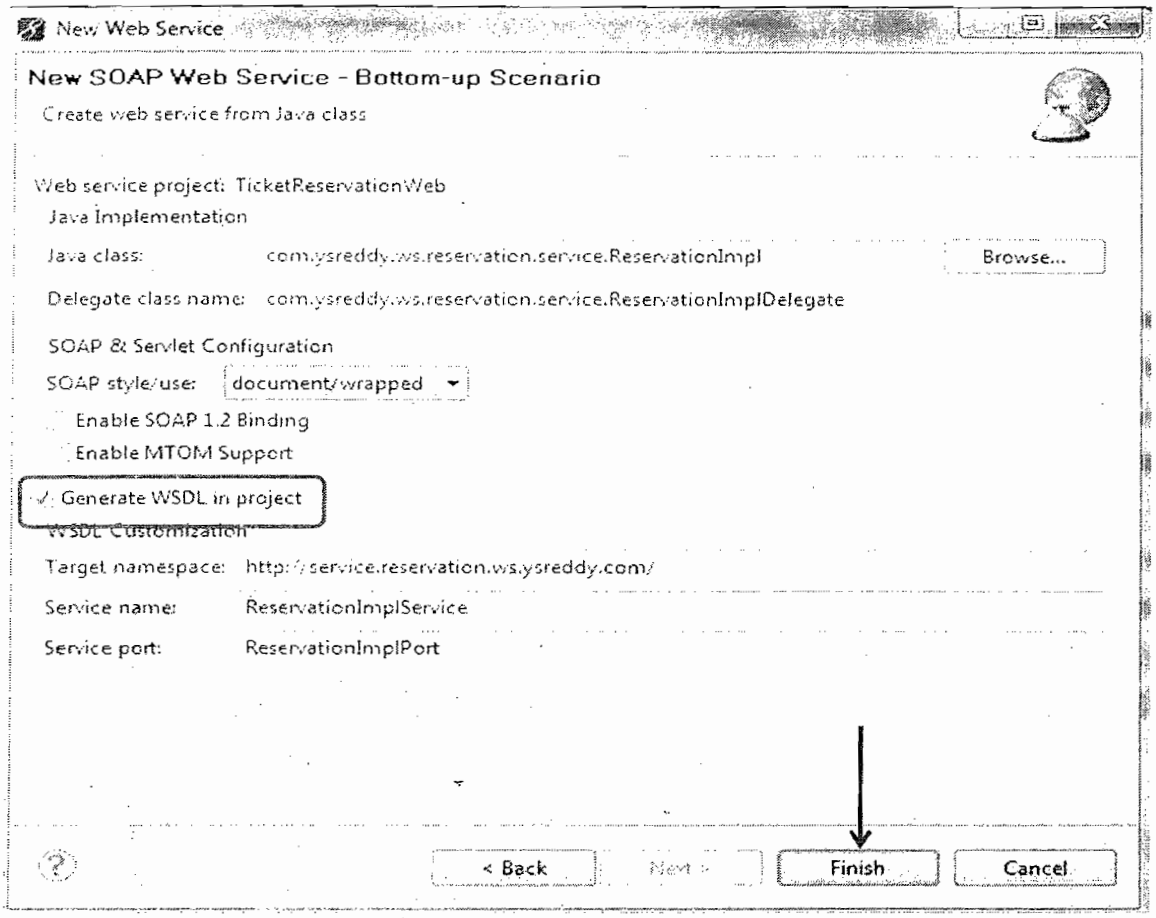
Step 3: Now we have to convert our service class (**ReservationImpl.java**) as a web service. To do that we start by clicking the **New Web Service** toolbar button:



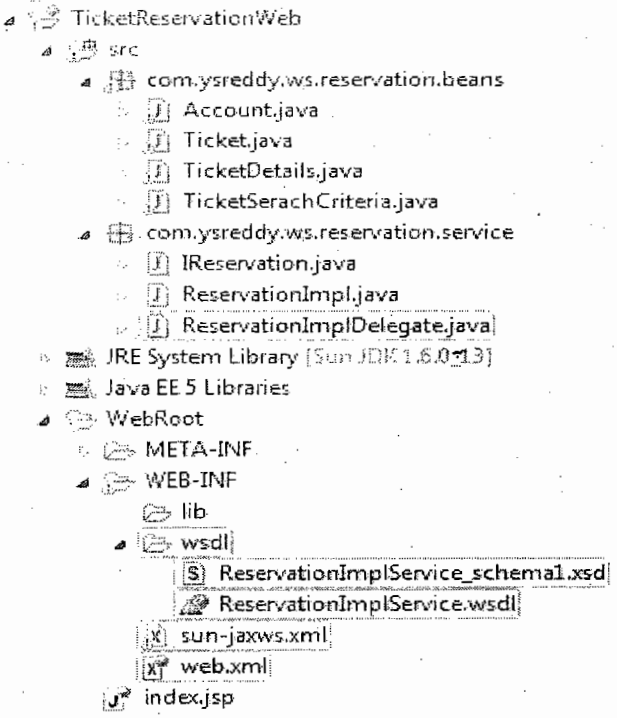
Step 4: In the next screen select marked options, and then click on "Next"



tep 5: In the next screen "**Browse**" and select our SEI implemented class, and select marked options and click **"Finish"**



Step 6: Now you can find the following server side artifacts, and In **web.xml**, we can find some configurations.



sun-jaxws.xml

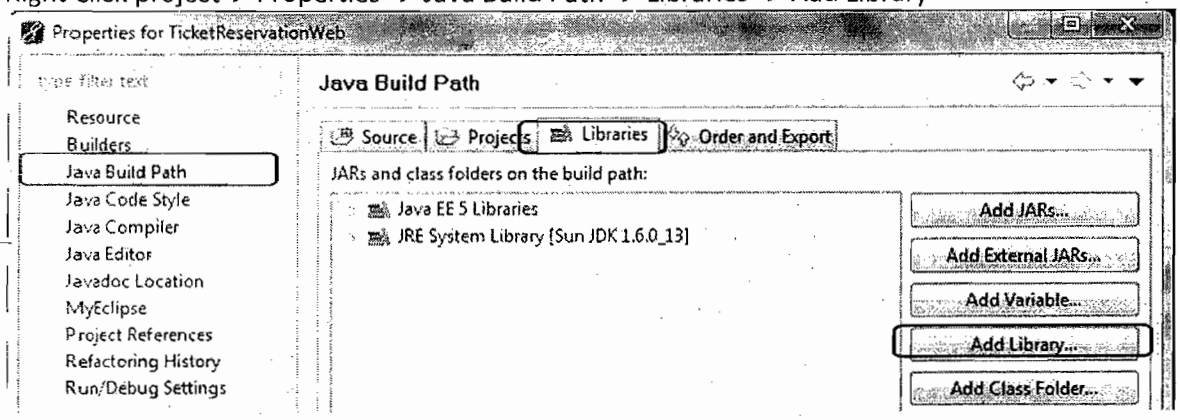
```
1. <?xml version = "1.0"?>
2. <endpoints version="2.0"
3.     xmlns="http://java.sun.com/xml/ns/jax-ws/ri/runtime">
4.     <endpoint name="ReservationImplPort"
5.           implementation="com.ysreddy.ws.reservation.service.ReservationImplDelegate"
6.           url-pattern="/ReservationImplPort">
7.     </endpoint>
8. </endpoints>
```

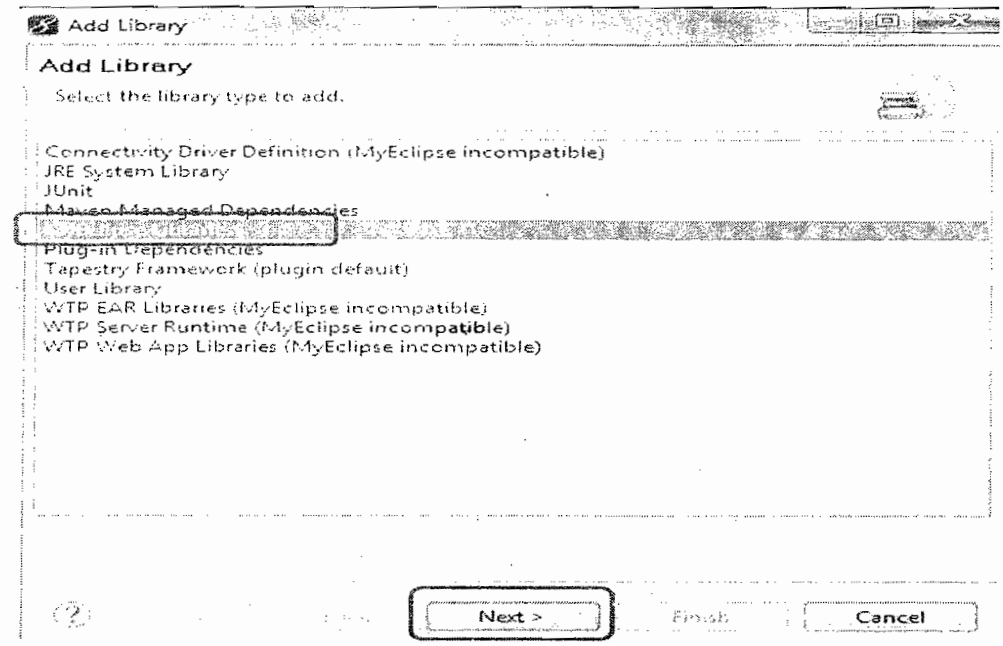
web.xml

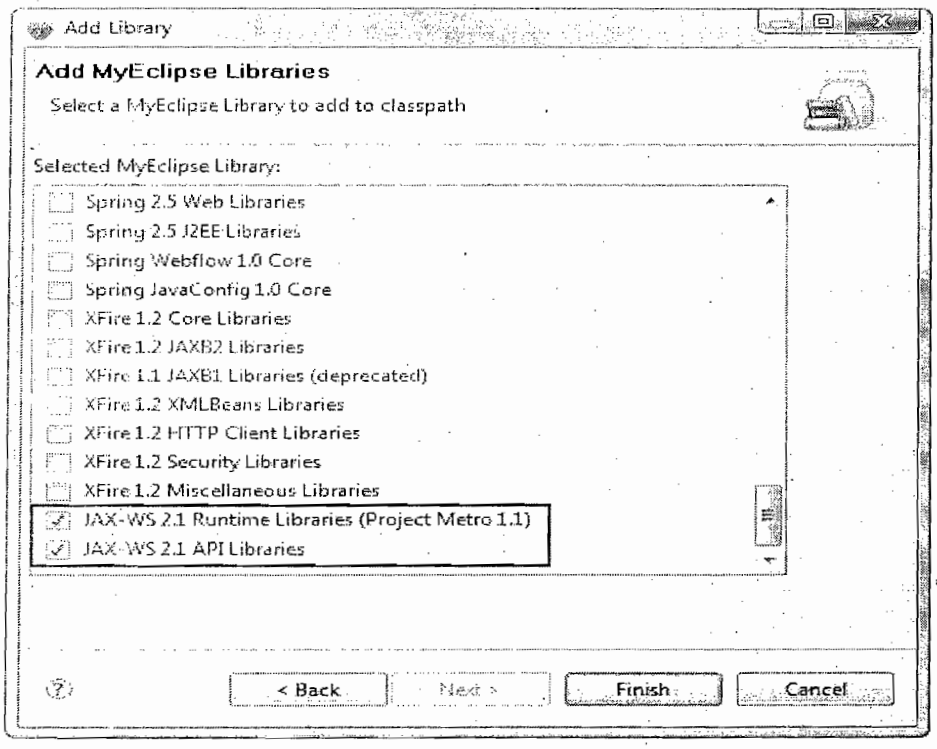
```
1. <?xml version="1.0" encoding="UTF-8"?>
2. <web-app version="2.5" xmlns="http://java.sun.com/xml/ns/javaee"
3.     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4.     xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
5.     http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd">
6.     <servlet>
7.           <description>
8.           JAX-WS endpoint - ReservationImplService
9.     </description>
10.              <display-name>ReservationImplService</display-name>
11.              <servlet-name>ReservationImplService</servlet-name>
12.              <servlet-class>com.sun.xml.ws.transport.http.servlet.WSServlet
13.               </servlet-class>
14.              <load-on-startup>1</load-on-startup>
15.          </servlet>
16.          <servlet-mapping>
17.              <servlet-name>ReservationImplService</servlet-name>
18.              <url-pattern>/ReservationImplPort</url-pattern>
19.          </servlet-mapping>
20.          <welcome-file-list>
21.              <welcome-file>index.jsp</welcome-file>
22.          </welcome-file-list>
23.          <listener>
24.              <listener-class>
25.                   com.sun.xml.ws.transport.http.servlet.WSServletContextListener
26.               </listener-class>
27.          </listener>
28.   </web-app>
```

Step 7: Now add jar files of JAX-WS

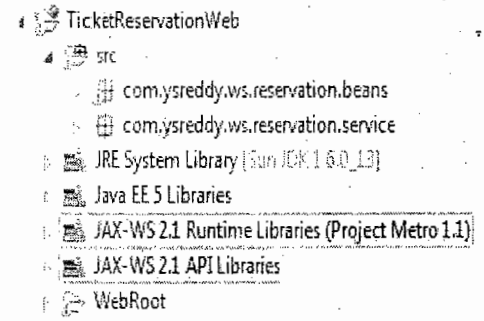
Right click project→ Properties → Java Build Path → Libraries → Add Library







Now click on "Finish" button, then you can find, JAX-WS libraries in our project as follows.



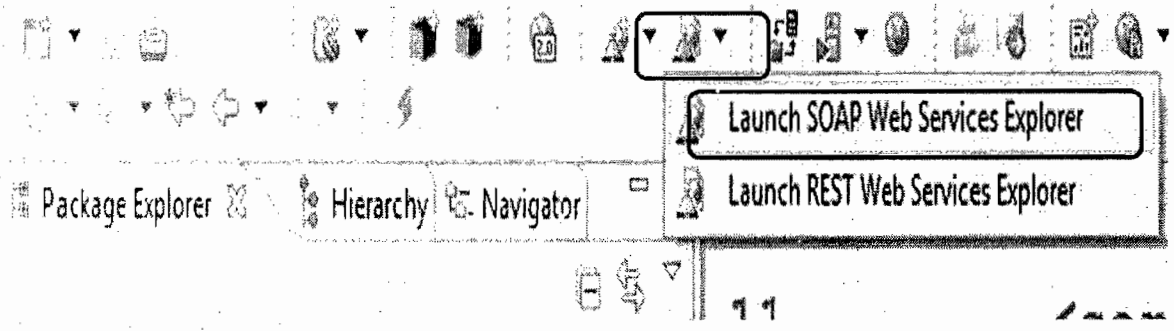
Step 8: Now deploy our project into server, and try to access the WSDL with following URL
http://localhost:8086/TicketReservationWeb/ReservationImplPort?wsdl

This XML file does not appear to have any style information associated with it. The document tree is shown below.

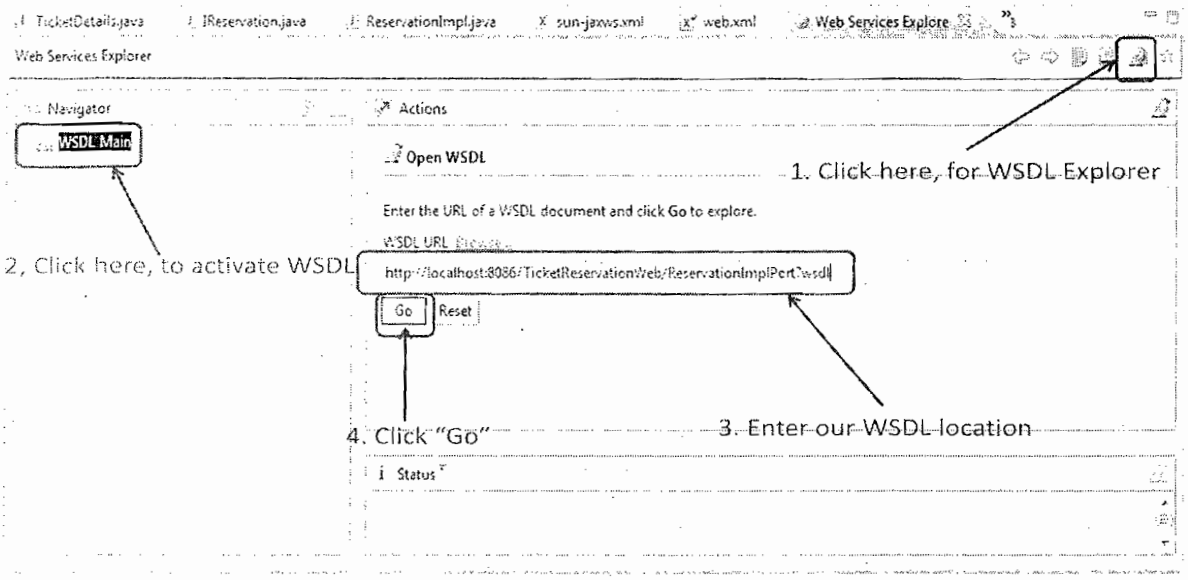

```
<!--
  Published by JAX-WS RI at http://jax-ws.dev.java.net. RI's version is JAX-WS RI 2.1.3-hudson-390-.
-->
<!--
  Generated by JAX-WS RI at http://jax-ws.dev.java.net. RI's version is JAX-WS RI 2.1.3-hudson-390-.
-->
<definitions xmlns="http://schemas.xmlsoap.org/wsdl/" xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/" xmlns:tns="http://service.reservation.ws.yareddy.com/"
 xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 name="ReservationImplService" targetNamespace="http://service.reservation.ws.yareddy.com/">
 <types>
  <xsd:schema>
   <xsd:import namespace="http://service.reservation.ws.yareddy.com/" schemaLocation="http://localhost:8086/TicketReservationWeb/ReservationImplPort?xsd=1"/>
  </xsd:schema>
 </types>
 <message name="searchTicket">
  <part element="tns:searchTicket" name="parameters"/>
 </message>
 <message name="searchTicketResponse">
  <part element="tns:searchTicketResponse" name="parameters"/>
 </message>
 <message name="buyTicket">
  <part element="tns:buyTicket" name="parameters"/>
 </message>
 <message name="buyTicketResponse">
  <part element="tns:buyTicketResponse" name="parameters"/>
 </message>
 <portType name="ReservationImplDelegate">
  <operation name="searchTicket">
   <input message="tns:searchTicket"/>
   <output message="tns:searchTicketResponse"/>
  </operation>
  <operation name="buyTicket">
   <input message="tns:buyTicket"/>
   <output message="tns:buyTicketResponse"/>
  </operation>
 </portType>
 <binding name="ReservationImplPortBinding" type="tns:ReservationImplDelegate">
  <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
```


Step 9: Now we can test our service with SOAP UI

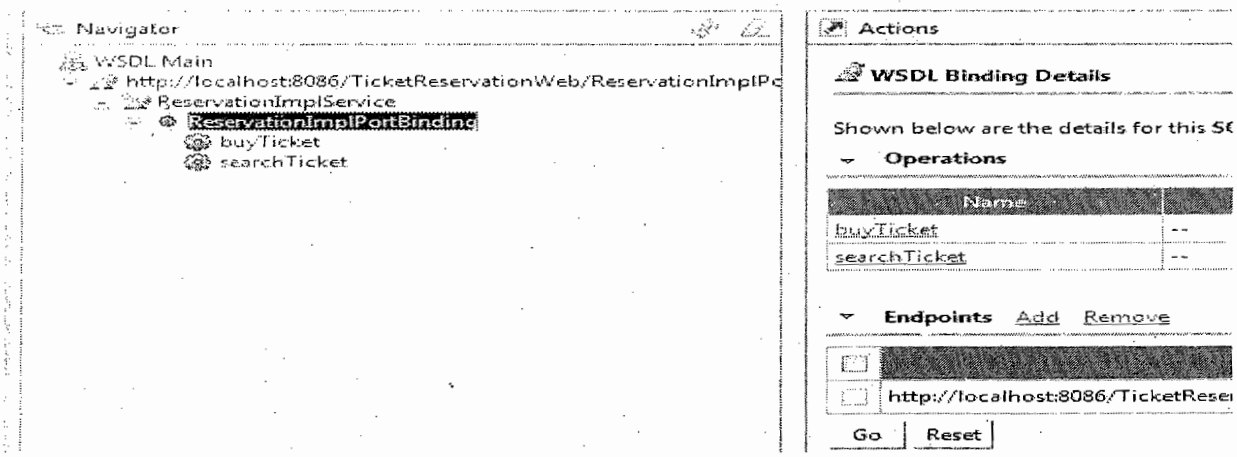
Step 10 : Myeclipse has given **Web service Explorer** to test web services graphically. Select the below marked options



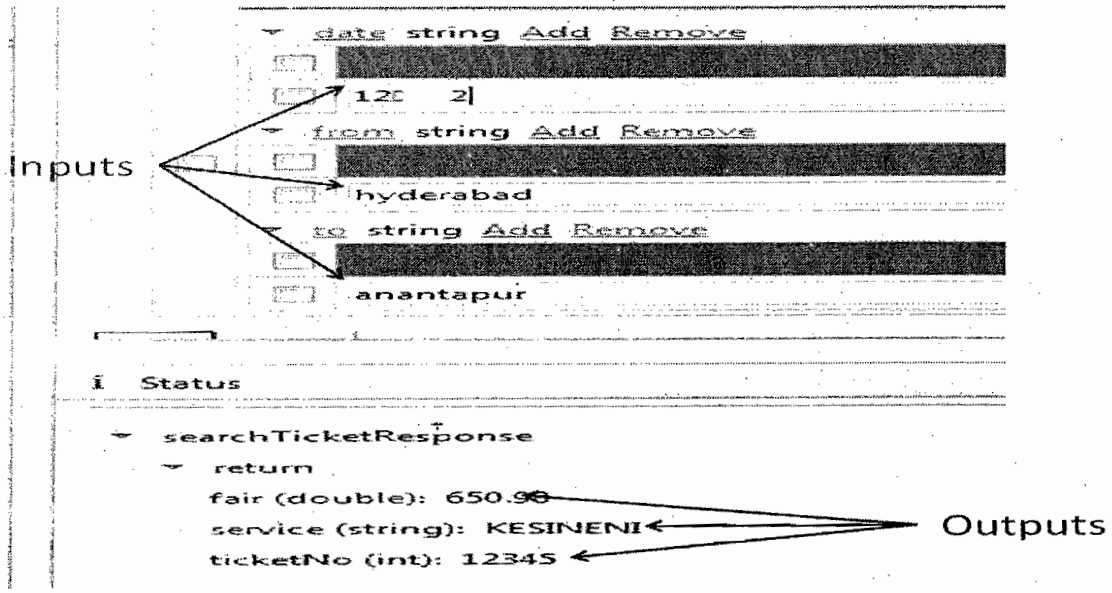
Now you will get the following window, in that select the specified options.



Now you will get the following screen

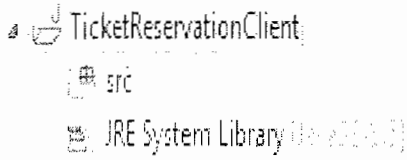


Now to test select **"searchTicket"** option, and provide the required inputs, then will get the response.

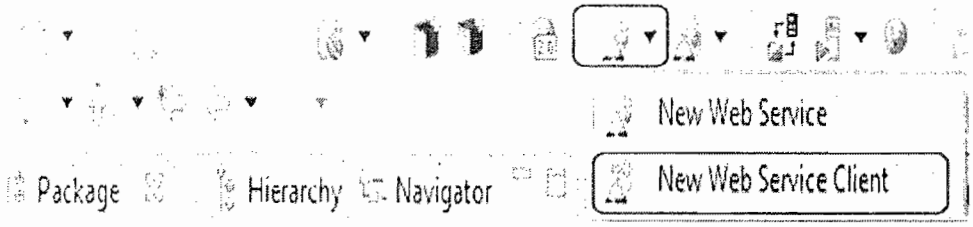


Creating a Client for the Web Service

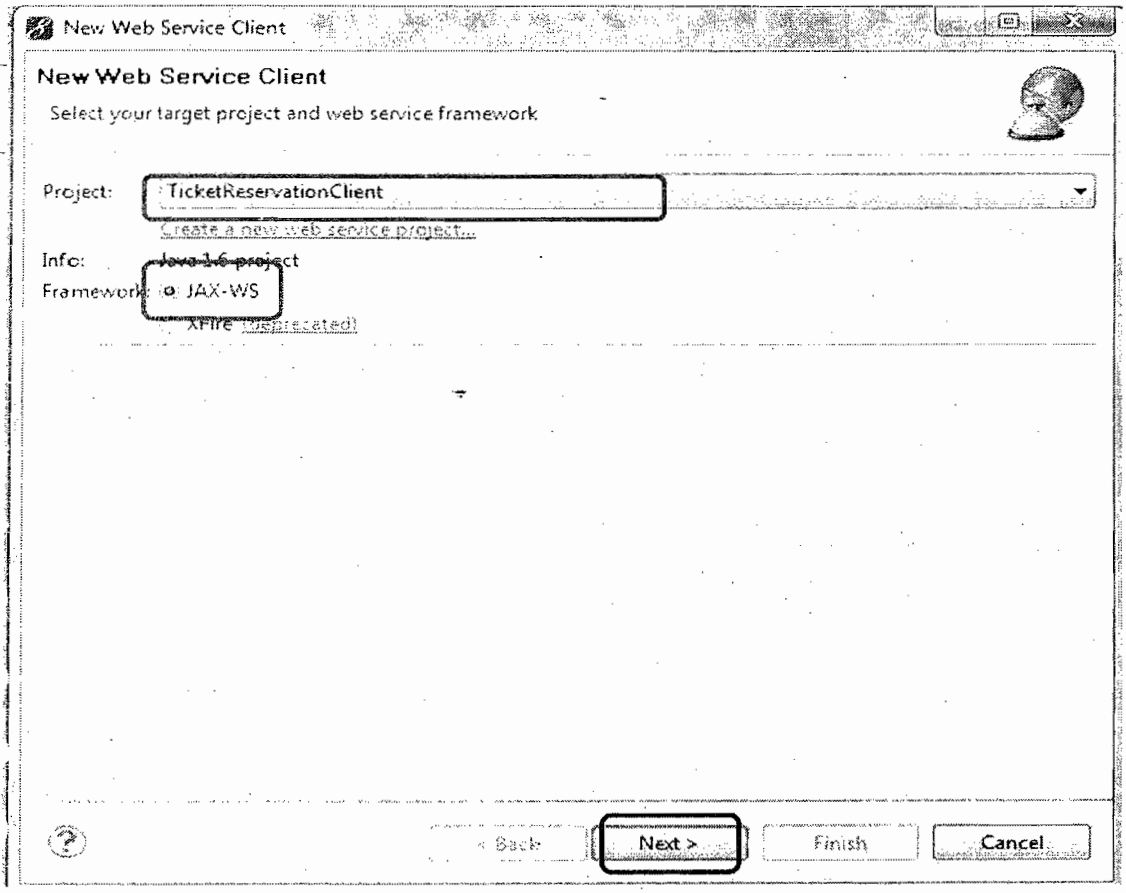
Step 1: create a "**java project**" with any name like "**TicketReservationClient**"



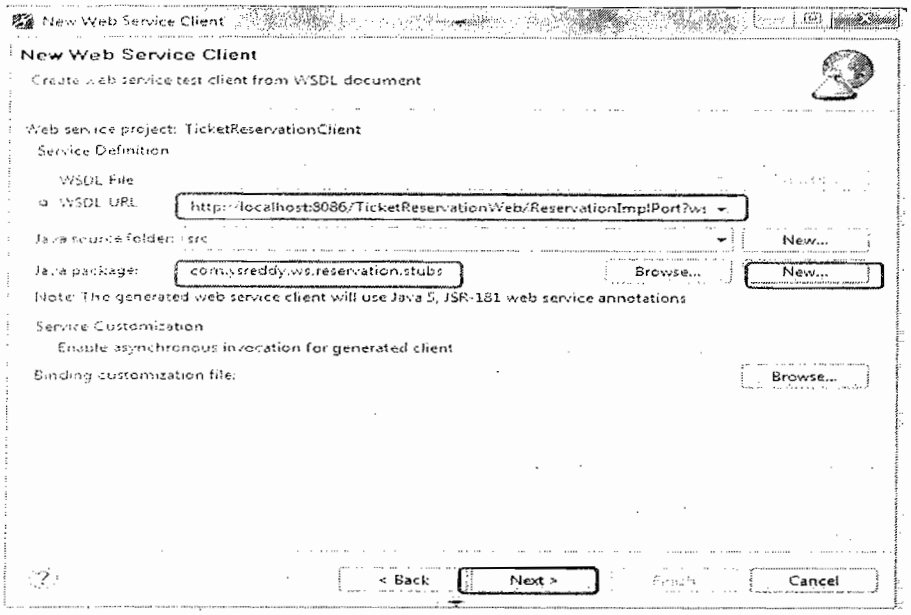
Step 2: Now select **"New Web Service Client"** from the web service toolbar menu

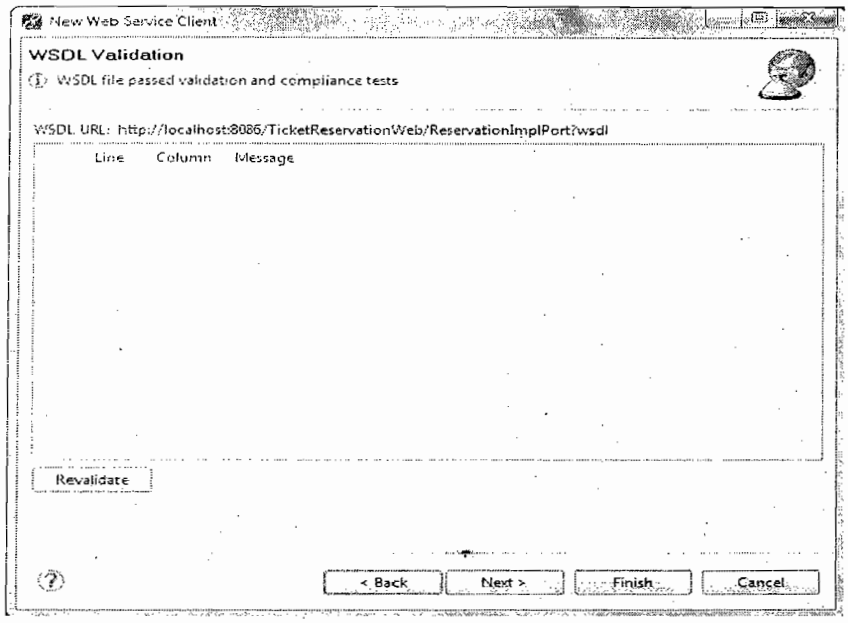


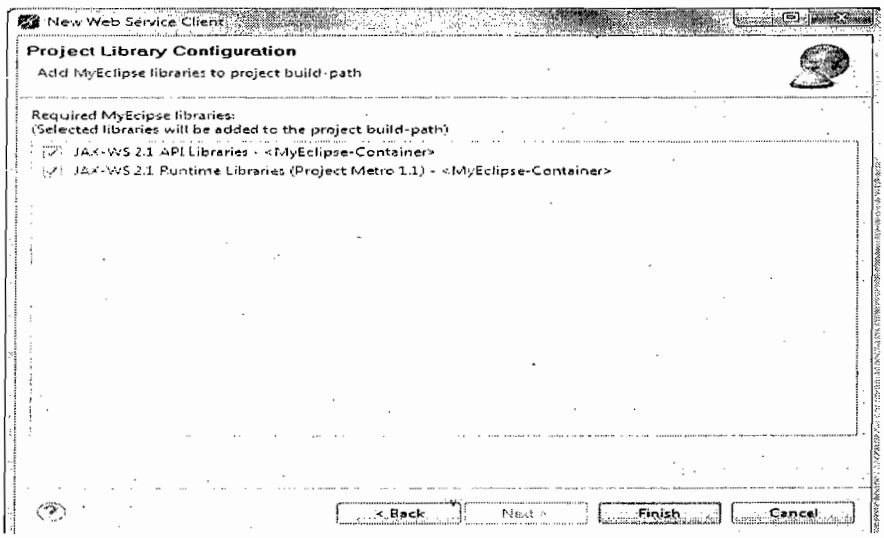
Now select Marked options, and then click on "Next"



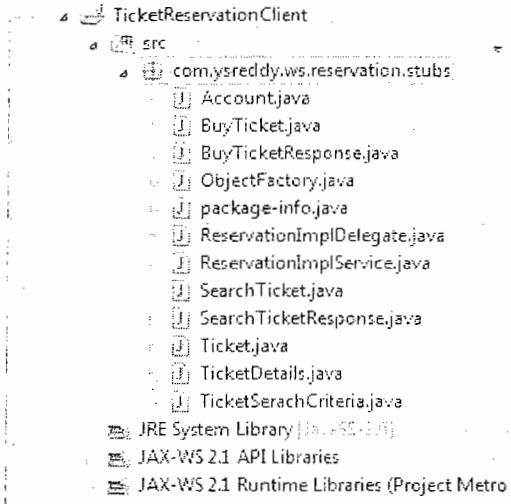
Now provide "WSDL" url, and create a package to place client side artifacts, then click on "Next"


New Web Service Client
New Web Service Client
Create a web service test client from WSDL document
Web service project: TicketReservationClient
Service Definition
WSDL File
WSDL URL
http://localhost:8086/TicketReservationWeb/ReservationImplPort?ws
Java source folder: src
New...
Java package:
com.sysreddy.ws.reservation.stubs
Browse...
New...
Note: The generated web service client will use Java 5, JSR-181 web service annotations
Service Customization
Enable asynchronous invocation for generated client
Binding customization file:
Browse...
< Back
Next >
Cancel


New Web Service Client
WSDL Validation
WSDL file passed validation and compliance tests
WSDL URL: http://localhost:8086/TicketReservationWeb/ReservationImplPort?wsdl
Line
Column
Message
Revalidate
< Back
Next >
Finish
Cancel


New Web Service Client
Project Library Configuration
Add MyEclipse libraries to project build-path
Required MyEclipse libraries:
(Selected libraries will be added to the project build-path)
JAX-WS 2.1 API Libraries - <MyEclipse-Container>
JAX-WS 2.1 Runtime Libraries (Project Metro 1.1) - <MyEclipse-Container>
< Back
Next >
Finish
Cancel

Now click on "Finish" button, then you can find client side artifacts in our project.



Step 3: Now write Client program

ReservationClient.java

```
1. package com.ysreddy.ws.reservation.stubs.client;
2. import com.ysreddy.ws.reservation.stubs.Account;
3. import com.ysreddy.ws.reservation.stubs.ReservationImplDelegate;
4. import com.ysreddy.ws.reservation.stubs.ReservationImplService;
5. import com.ysreddy.ws.reservation.stubs.Ticket;
6. import com.ysreddy.ws.reservation.stubs.TicketDetails;
7. import com.ysreddy.ws.reservation.stubs.TicketSerachCriteria;
8.
9. public class ReservationClient {
10.         public static void main(String[] args) {
11.               ReservationImplService  service = new ReservationImplService();
12.               ReservationImplDelegate delegate = service.getReservationImplPort();
13.
14.               System.out.println("...Searching Ticket...");
15.               TicketSerachCriteria criteria = new TicketSerachCriteria();
16.               criteria.setFrom("Hyderabad");
17.               criteria.setTo("Anantapur");
18.               criteria.setDate("22/03/2012");
19.
20.               Ticket ticket = delegate.searchTicket(criteria);
21.               System.out.println("Ticket NO:"+ticket.getTicketNo());
22.               System.out.println("Service :"+ticket.getFair());
23.               System.out.println("Fair :"+ticket.getFair());
24.
25.               System.out.println("...Buy ticket...");
26.               Account account = new Account();
27.               account.setAccno(1001);
28.               account.setName("Sekhar");
29.               TicketDetails details = delegate.buyTicket(account);
30.
31.               System.out.println("Seat No : "+details.getSeatNo());
32.               System.out.println("Date : "+details.getDate());
33.               System.out.println("Time : "+details.getTime());
34.               System.out.println("Bording Point : "+details.getBoardingPoint());
35.         }
36. }
```

JAX-RS

- JAX-RS stands for Java API for XML RESTful Web services
- REST is an architectural style which is based on web-standards and the HTTP protocol.
- In a REST based architecture everything is a resource. A resource is accessed via a common interface based on the HTTP standard methods. In an REST architecture you typically have a REST server which provides access to the resources and a REST client which accesses and modify the REST resources.
- Every resource should support the HTTP common operations. Resources are identified by global ID's (which are typically URIs).
- REST allows that resources have different representations, e.g. text, xml, json etc. The rest client can ask for specific representation via the HTTP protocol (Content Negotiation).

API	Implementation
JAX-RS	1. Jersey 2. RESTEasy(from JBoss) 3. Restlet 4. Apache CXF

HTTP methods

The HTTP standards methods which are typical used in REST are PUT, GET, POST, DELETE.

- GET defines a reading access of the resource without side-effects. The resource is never changed via a GET request, e.g. the request has no side effects (idempotent).
- PUT creates a new resource, must also be idempotent.
- DELETE removes the resources. The operations are idempotent, they can get repeated without leading to different results.
- POST updates an existing resource or creates a new resource.

RESTFul webservices

A RESTFul webservices is based on the HTTP methods and the concept of REST. It typically defines the base URI for the services, the MIME-types its supports (XML, Text, JSON, user-defined,..) and the set of operations (POST, GET, PUT, DELETE) which are supported. JAX-RS supports the creation of XML and JSON via JAXB.

Java, REST and Jersey

Java defines standard REST support via JAX-RS (The Java API for RESTful Web Services) in JSR 311.JAX-RS uses annotations to define the REST relevance of classes.

Via your "web.xml" you will register a servlet provided by Jersey and also define the path under which your REST web application will be available. The base URL of this servlet is:

```
http://your_domain:port/display-name/url-pattern/path_from_rest_class
```

This servlet which analyze the incoming HTTP request and select the correct class and method to respond to this request. This selection is based on annotation in the class and methods.

The most important annotations in JAX-RS are the following.

Table 1. Sample Table

Annotation	Description
@PATH(your_path)	Sets the path to base URL + /your_path. The base URL is based on your application name, tl servlet and the URL pattern from the web.xml" configuration file.
@POST	Indicates that the following method will answer to a HTTP POST request
@GET	Indicates that the following method will answer to a HTTP GET request
@PUT	Indicates that the following method will answer to a HTTP PUT request
@DELETE	Indicates that the following method will answer to a HTTP DELETE request
@Produces(MediaType.TEXT_PLAIN [, more-types])	@Produces defines which MIME type is delivered by a method annotated with @GET. In th example text ("text/plain") is produced. Other examples would be "application/xml" or "application/json".

Annotation	Description
@Consumes(type [, more-types])	@Consumes defines which MIME type is consumed by this method.
@PathParam	Used to inject values from the URL into a method parameter. This way you inject for exam the ID of a resource into the method to get the correct object.

The complete path to a resource is therefore based on the base URL and the @PATh annotation in your class.

`http://your_domain:port/display-name/url-pattern/path_from_rest_class`

Jersey is the reference implementation for this specification. Jersey contains basically a REST server and a REST client. The core client is mainly available for testing and provides a library to communicate with the server.

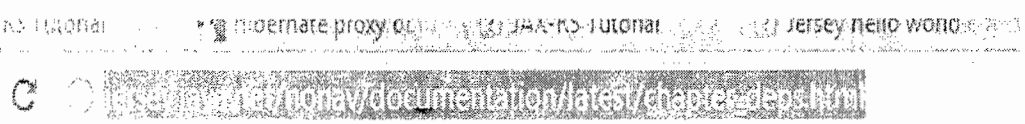
A REST web application consists out of data classes (resources) and services. These two types are typically maintained in different packages as the Jersey servlet will be instructed via the "web.xml"to scan certain packages for data classes.

Q.) Develop simple Restful web service, using jersey implementation?

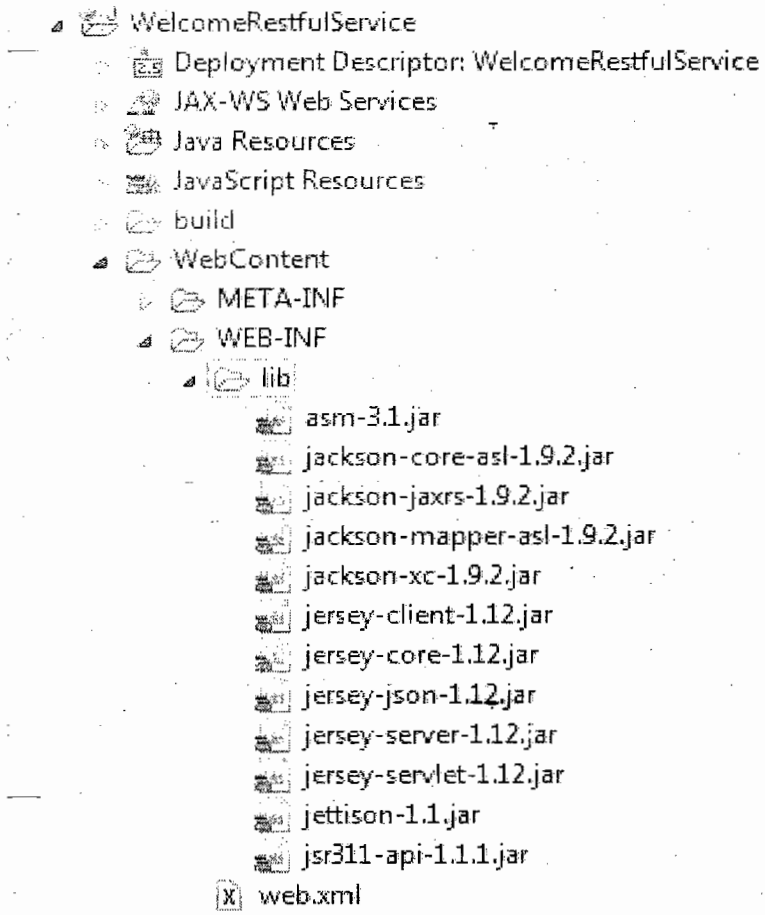
Step 1: create a "Dynamic Web Project" with any name like "WelcomeRestfulService"

Step 2: Add "jersey" implementation jar files by downloading from the following location.

http://jersey.java.net/nonav/documentation/latest/chapter_deps.html



Now you will get "**jersey-archive-1.12.zip**" file, then extract it, then go to lib folder, copy all jar files, and paste into our project "**WebContent/WEB-INF/lib**". Now the project structure looks like as follows...



Step 3: Now create a Restful services with the help of annotations

WelcomeService.java

```
1. package com.ysreddy.restful.service;
2.
3. import javax.ws.rs.GET;
4. import javax.ws.rs.Path;
5. import javax.ws.rs.PathParam;
6.
7. @Path("/welcome")
8. public class WelcomeService {
9.
10.         @GET
11.         @Path("/{name}")
12.         public String getWelcomeMsg(@PathParam("name")String name){
13.               return "Welcome : "+name;
14.         }
15.   }
```

Step 4: In `web.xml`, register "`com.sun.jersey.spi.container.servlet.ServletContainer`", and puts your Jersey service folder under "**init-param**", "`com.sun.jersey.config.property.packages`".

web.xml

```
1. <?xml version="1.0" encoding="UTF-8"?>
2. <web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3. xmlns="http://java.sun.com/xml/ns/javaee"
4. xmlns:web="http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
5. xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
6. http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd" id="WebApp_ID" version="2.5">
7.
8.     <display-name>WelcomeRestfulService</display-name>
9.
10.         <servlet>
11.               <servlet-name>jersey-serlvet</servlet-name>
12.               <servlet-class>
13.                         com.sun.jersey.spi.container.servlet.ServletContainer
14.                   </servlet-class>
15.               <init-param>
16.                    <param-name>com.sun.jersey.config.property.packages</param-name>
17.                    <param-value> com.ysreddy.restful.service</param-value>
18.               </init-param>
19.               <load-on-startup>1</load-on-startup>
20.         </servlet>
21.
22.         <servlet-mapping>
23.               <servlet-name>jersey-serlvet</servlet-name>
24.               <url-pattern>/rest/*</url-pattern>
25.         </servlet-mapping>
26.
27.   </web-app>
```

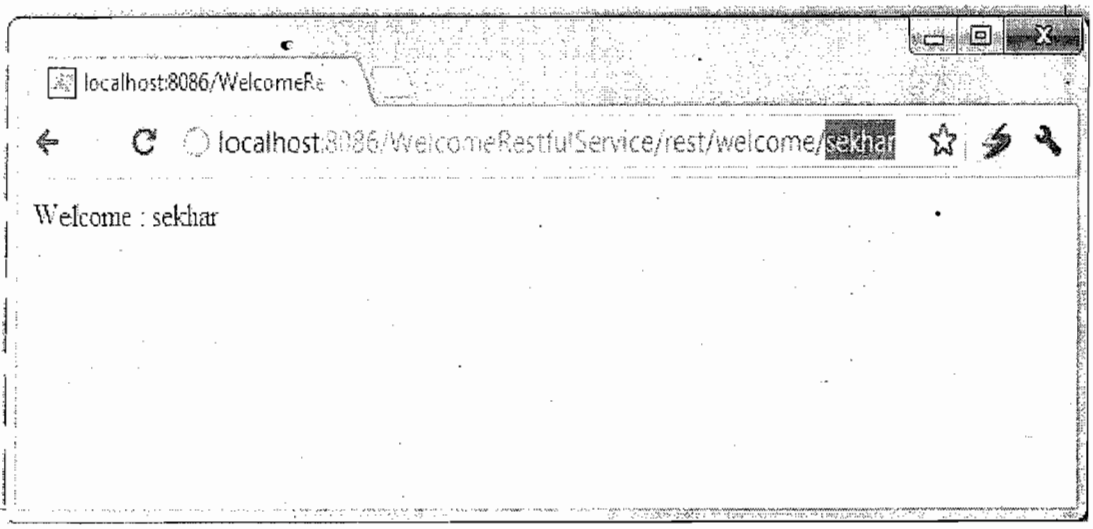
Now the final project structure looks like as follows.

Step 5: Now open the browser and send the request to service with the following url

In this example, web request from "**projectURL/rest/welcome/**" will match to "**WelcomeService**", via `@Path("/welcome")`.

And the "**{any values}**" from "**projectURL/rest/welcome/{any values}**" will match to parameter annotated with `@PathParam`.

http://localhost:8086/WelcomeRestfulService/rest/welcome/sekhar

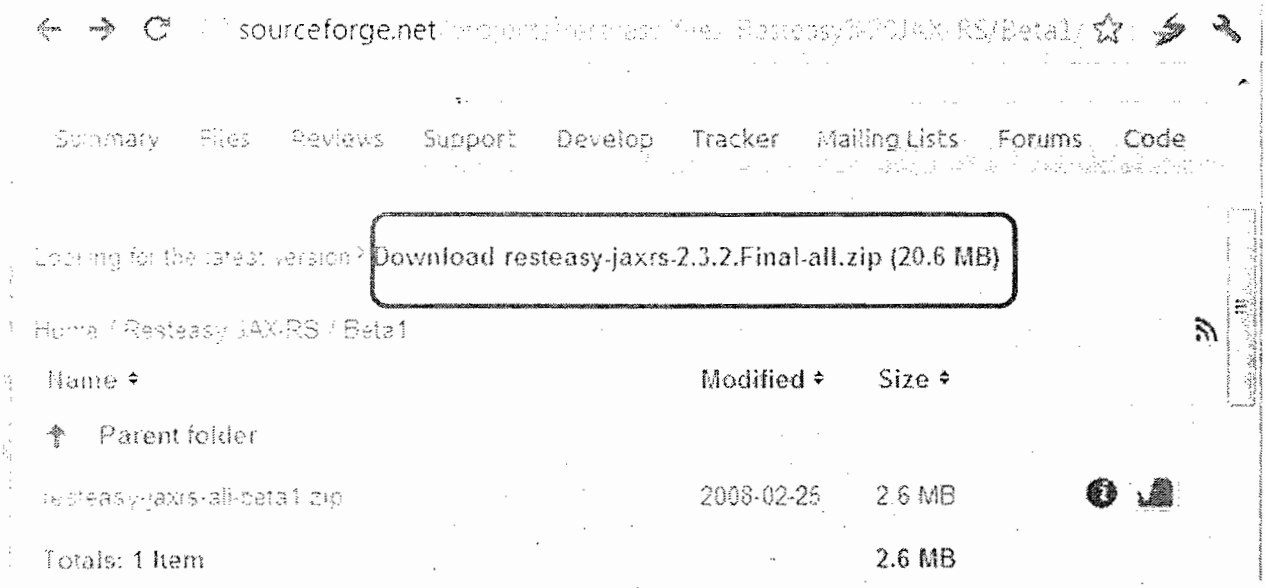


2.) Develop simple Restful web service, using RESTEasy implementation?

Step 1: create a **"Dynamic Web Project"** with any name like **"RestEasySevice"**

Step 2: Now download the RESTEasy implementation jars from the following link

http://sourceforge.net/projects/resteasy/files/Resteasy%20JAX-RS/Beta1/



Now you can find the downloaded jar file as **"resteasy-jaxrs-2.3.2.Final-all.zip"** and extract it and we can find the "lib" folder in the extraction. From there copy all jar files and paste into our **"WebContetn/WEB-INF/lib"** folder

Step 3: Now create any package and develop web service class...

WelcomeService.java

```
1. package com.ysreddy.restful.service;
2.
3. import javax.ws.rs.GET;
4. import javax.ws.rs.Path;
5. import javax.ws.rs.PathParam;
6.
7. @Path("/welcome")
8. public class WelcomeService {
9.
10.         @GET
11.         @Path("/{name}")
12.         public String getWelcomeMsg(@PathParam("name")String name){
13.               return "Welcome : "+name;
14.         }
15.   }
```

Step 4: Now, configure listener and servlet to support RESTEasy in web.xml

web.xml

```
1. <web-app id="WebApp_ID" version="2.4"
2.     xmlns="http://java.sun.com/xml/ns/j2ee"
3.     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

```
4.    xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
5.    http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">
6.    <display-name>Restful Web Application</display-name>
7.
8.    <!-- Auto scan REST service -->
9.          <context-param>
10.               <param-name>resteasy.scan</param-name>
11.               <param-value>true</param-value>
12.         </context-param>
13.
14.         <!-- this need same with resteasy servlet url-pattern -->
15.         <context-param>
16.               <param-name>resteasy.servlet.mapping.prefix</param-name>
17.               <param-value>/rest</param-value>
18.         </context-param>
19.
20.         <listener>
21.               <listener-class>
22.                     org.jboss.resteasy.plugins.server.servlet.ResteasyBootstrap
23.               </listener-class>
24.         </listener>
25.
26.         <servlet>
27.               <servlet-name>resteasy-servlet</servlet-name>
28.               <servlet-class>
29.                     org.jboss.resteasy.plugins.server.servlet.HttpServletDispatcher
30.               </servlet-class>
31.         </servlet>
32.
33.         <servlet-mapping>
34.               <servlet-name>resteasy-servlet</servlet-name>
35.               <url-pattern>/rest/*</url-pattern>
36.         </servlet-mapping>
37.
38.   </web-app>
```

Note: You need to set the "**resteasy.servlet.mapping.prefix**" if your servlet-mapping for the resteasy servlet has a url-pattern other than "/*".

n above example, the resteasy servlet url-pattern is "**/rest/***", so you have to set the "resteasy.servlet.mapping.prefix" to "**/rest**" as well, otherwise, you will hit resource not found error message.

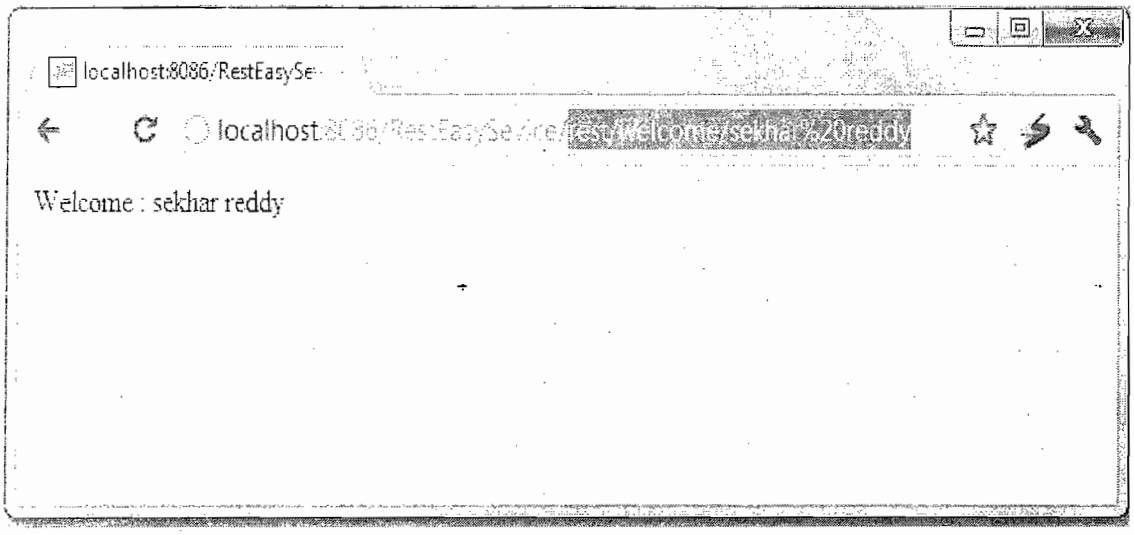
Note: Remember to set "**resteasy.scan**" to true, so that RESTEasy will find and register your REST service automatically.

Step 5: Now deploy the above project into server, and open the browser and send the request to the developed web service.

http://localhost:8086/RestEasySevice/rest/welcome/sekhar

n this example, web request from "**projectURL/rest/welcome/**" will match to "`WelcomeService`", and "**projectURL/rest/welcome/{any values}**" will match to **@PathParam** parameter.

http://localhost:8086/RestEasySevice/rest/welcome/sekhar%20reddy



Alternative REST Service Registration

In above example, you are register REST service via "**ResteasyBootstrap**" listener. Here i show you another way. Create a class and extends `javax.ws.rs.core.Application`, and add your REST service manually.

MessageApplication.java

```
1. package com.ysreddy.restful.service;
2. import java.util.HashSet;
3. import java.util.Set;
4. import javax.ws.rs.core.Application;
5. public class MessageApplication extends Application {
6.     private Set<Object> singletons = new HashSet<Object>();
7.
8.     public MessageApplication() {
9.           singletons.add(new WelcomeService());
10.          }
11.
12.          @Override
```

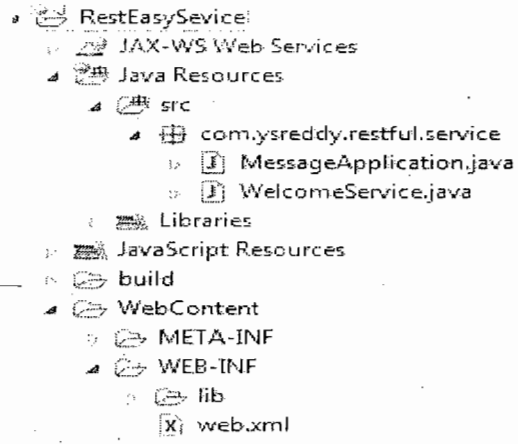
```
13.         public Set<Object> getSingletons() {
14.               return singletons;
15.         }
16.   }
```

File : web.xml , no more listener, configure your application class like below :

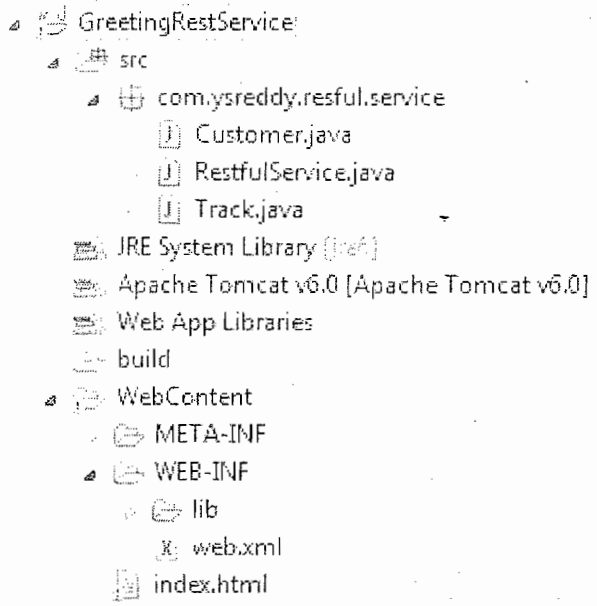
web.xml

```
1. <?xml version="1.0" encoding="UTF-8"?>
2. <web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3. xmlns="http://java.sun.com/xml/ns/javaee"
4. xmlns:web="http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
5. xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
6. http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd" id="WebApp_ID" version="2.5">
7. <display-name>Restful Web Application</display-name>
8.
9.    <context-param>
10.             <param-name>resteasy.servlet.mapping.prefix</param-name>
11.             <param-value>/rest</param-value>
12.       </context-param>
13.
14.       <servlet>
15.             <servlet-name>resteasy-servlet</servlet-name>
16.             <servlet-class>
17.                   org.jboss.resteasy.plugins.server.servlet.HttpServletDispatcher
18.               </servlet-class>
19.             <init-param>
20.                   <param-name>javax.ws.rs.Application</param-name>
21.                   <param-value>com.ysreddy.restful.service.MessageApplication
22.                    </param-value>
23.             </init-param>
24.       </servlet>
25.
26.       <servlet-mapping>
27.             <servlet-name>resteasy-servlet</servlet-name>
28.             <url-pattern>/rest/*</url-pattern>
29.       </servlet-mapping>
30.
31.
32.   </web-app>
```

Final structure of the project is



Q.) Develop a restful service where we can use of all possible annotations?



RestfulService.java

```
1. package com.ysreddy.resful.service;
2.
3. import java.io.File;
4. import java.util.List;
5.
6. import javax.ws.rs.Consumes;
7. import javax.ws.rs.DefaultValue;
8. import javax.ws.rs.FormParam;
9. import javax.ws.rs.GET;
10.    import javax.ws.rs.HeaderParam;
11.    import javax.ws.rs.MatrixParam;
12.    import javax.ws.rs.POST;
13.    import javax.ws.rs.Path;
14.    import javax.ws.rs.PathParam;
15.    import javax.ws.rs.Produces;
16.    import javax.ws.rs.QueryParam;
17.    import javax.ws.rs.core.Context;
18.    import javax.ws.rs.core.HttpHeaders;
19.    import javax.ws.rs.core.MediaType;
20.    import javax.ws.rs.core.Response;
21.    import javax.ws.rs.core.Response.ResponseBuilder;
22.    import javax.ws.rs.core.UriInfo;
23.
24.
25.    //Sets the path to base URL + /common
26.    // http://www.mkyong.com/tutorials/jax-rs-tutorials/
27.    @Path("/common")
28.    public class RestfulService {
29.
30.        // This method is called if TEXT_PLAIN is request
31.    //    http://localhost:8086/GreetingRestService/rest/common/2012/03/03
32.
```

```
33.         @GET
34.         @Path("{year}/{month}/{day}")
35.         public String getUserHistory(
36.                     @PathParam("year") int year,
37.                     @PathParam("month") int month,
38.                     @PathParam("day") int day) {
39.
40.            String date = year + "/" + month + "/" + day;
41.
42.            return "Date is " + date;
43.
44.         }
45.
46.   //    http://localhost:8086/GreetingRestService/rest/common/query?from=10
47.         &to=20&orderBy=abc&orderBy=xyz
48.
49.         @GET
50.         @Path("/query")
51.         public Response getUsers(
52.               @QueryParam("from") int from,
53.               @QueryParam("to") int to,
54.               @QueryParam("orderBy") List<String> orderBy) {
55.
56.               String result = "getUsers is called, from : " + from + ", to : " + to
57.                     + ", orderBy" + orderBy.toString();
58.               return Response.status(200).entity(result).build();
59.         }
60.
61.   //    http://localhost:8086/GreetingRestService/rest/common/query?from=10
62.         &to=20&orderBy=abc&orderBy=xyz
63.
64.         @GET
65.         @Path("/context")
66.         public Response getUsers(@Context UriInfo info) {
67.
68.               String from = info.getQueryParameters().getFirst("from");
69.               String to = info.getQueryParameters().getFirst("to");
70.               List<String> orderBy = info.getQueryParameters().get("orderBy");
71.               String result = "getUsers is called, from : " + from + ", to : " + to
72.                           + ", orderBy" + orderBy.toString();
73.               return Response.status(200).entity(result).build();
74.
75.         }
76.
77.   //    http://localhost:8086/GreetingRestService/rest/common/defaultValue
78.
79.         @GET
80.         @Path("/defaultValue")
81.         public Response getUsersDefault(
82.               @DefaultValue("1000") @QueryParam("from") int from,
83.               @DefaultValue("999")@QueryParam("to") int to,
84.               @DefaultValue("name") @QueryParam("orderBy") List<String> orderBy) {
85.
86.               String result = "getUsers is called, from : " + from + ", to : " + to
87.                           + ", orderBy" + orderBy.toString();
```

```
88.                 return Response.status(200).entity(result).build();
89.
90.         }
91.
92.   //    http://localhost:8086/GreetingRestService/rest/common/2012;author=sekhar;
93.         country=india
94.
95.         @GET
96.         @Path("/matrixParam")
97.         public Response getMatrixParam(     @MatrixParam("author") String author,
98.                                             @MatrixParam("country") String country) {
99.
100.        String result="getBooks is called, author:"+author+ ", country : " + country;
101.              return Response.status(200).entity(result).build();
102.
103.        }
104.
105.        // Form params send through UserForm
106.
107.        @POST
108.        @Path("/add")
109.        public Response addUser(
110.              @FormParam("name") String name,
111.              @FormParam("age") int age) {
112.
113.              String result = "addUser is called, name : " + name + ", age : " + age;
114.              return Response.status(200).entity(result).build();
115.
116.        }
117.
118.        //get HTTP headers in JAX-RS
119.  //    http://localhost:8086/GreetingRestService/rest/common/getHeaderParam
120.
121.        @GET
122.        @Path("/getHeaderParam")
123.        public Response getHeaderParam(@HeaderParam("user-agent") String userAgent) {
124.              String result = "getHeaderParam is called, userAgent : " + userAgent;
125.              return Response.status(200).entity(result).build();
126.
127.        }
128.
129.  //    http://localhost:8086/GreetingRestService/rest/common/getContextHttpHeaders
130.
131.        @GET
132.        @Path("/getContextHttpHeaders")
133.        public Response getContextHttpHeaders(@Context HttpHeaders headers) {
134.
135.              String userAgent = headers.getRequestHeader("user-agent").get(0);
136.              String result = "addUser is called, userAgent : " + userAgent;
137.              return Response.status(200).entity(result).build();
138.
139.        }
140.
141.  //http://localhost:8086/GreetingRestService/rest/common/getAllContextHttpHeaders
142.
```

```
143.        @GET
144.        @Path("/getAllContextHttpHeaders")
145.        public Response getAllContextHttpHeaders(@Context HttpHeaders headers) {
146.
147.              String userAgent = "Sekhar : \n";
148.              for(String header : headers.getRequestHeaders().keySet()){
149.                    String userAgentValue = headers.getRequestHeader(header).get(0);
150.                 userAgent = userAgent + "\t" +header + "\t"+ userAgentValue+ "\n" ;
151.              }
152.              String result = "addUser is called, userAgent : " + userAgent;
153.              return Response.status(200).entity(result).build();
154.
155.        }
156.
157.        private static final String FILE_PATH = "E:\\downloads\\Address.txt";
158.  //    http://localhost:8086/GreetingRestService/rest/common/getFile
159.
160.        @GET
161.        @Path("/getFile")
162.        @Produces("text/plain")
163.        public Response getFile() {
164.
165.              File file = new File(FILE_PATH);
166.              ResponseBuilder response = Response.ok((Object) file);
167.              response.header("Content-Disposition", "attachment;
168.                                          filename=\"file_from_server.txt\"");
169.              return response.build();
170.
171.        }
172.
173.        private static final String FILE_PATH_IMAGE = "E:\\downloads\\Penguins.jpg";
174.  //    http://localhost:8086/GreetingRestService/rest/common/getImageFile
175.
176.        @GET
177.        @Path("/getImageFile")
178.        @Produces("image/jpg")
179.        public Response getImageFile() {
180.
181.              File file = new File(FILE_PATH_IMAGE);
182.              ResponseBuilder response = Response.ok((Object) file);
183.              response.header("Content-Disposition", "attachment;
184.                                                filename=image_from_server.jpg");
185.              return response.build();
186.
187.        }
188.
189.        private static final String FILE_PATH_PDF = "E:\\downloads\\http1.1.pdf";
190.  //    http://localhost:8086/GreetingRestService/rest/common/getPdfFile
191.
192.        @GET
193.        @Path("/getPdfFile")
194.        @Produces("application/pdf")
195.        public Response getPdfFile() {
196.
197.              File file = new File(FILE_PATH_PDF);
```

```
198.              ResponseBuilder response = Response.ok((Object) file);
199.              response.header("Content-Disposition",
200.                                      "attachment; filename=new-android-book.pdf");
201.              return response.build();
202.
203.        }
204.
205.        private static final String FILE_PATH_EXCEL = "E:\\downloads\\details.xlsx";
206.  //    http://localhost:8086/GreetingRestService/rest/common/getExcelFile
207.
208.        @GET
209.        @Path("/getExcelFile")
210.        @Produces("application/vnd.ms-excel")
211.        public Response getExcelFile() {
212.
213.              File file = new File(FILE_PATH_EXCEL);
214.              ResponseBuilder response = Response.ok((Object) file);
215.              response.header("Content-Disposition",
216.                                       "attachment; filename=new-excel-file.xls");
217.              return response.build();
218.
219.        }
220.
221.  //    http://localhost:8086/GreetingRestService/rest/common/customer/1001/sekhar
222.
223.        @GET
224.        @Path("/customer/{pin}/{custName}")
225.        @Produces(MediaType.APPLICATION_XML)
226.        public Customer getCustomerInXML(@PathParam("pin") int pin ,
227.                                         @PathParam("custName") String custName) {
228.
229.              Customer customer = new Customer();
230.              customer.setName(custName);
231.              customer.setPin(pin);
232.              return customer;
233.
234.        }
235.
236.  //    http://localhost:8086/GreetingRestService/rest/common/sekhar
237.
238.     @GET
239.     @Path("/{name}")
240.     @Produces(MediaType.TEXT_PLAIN)
241.     public String sayPlainTextHello(@PathParam("name") String name1) {
242.         return "Hello Jersey : \t"+name1 + "\tWelcome";
243.     }
244.
245.  //    http://localhost:8086/GreetingRestService/rest/common/xml
246.
247.     // This method is called if XMLis request
248.
249.     @GET
250.     @Path("/xml")
251.     @Produces(MediaType.TEXT_XML)
252.     public String sayXMLHello() {
```

```
253.          return "<?xml version=\"1.0\"?>" + "<hello> Hello Jersey" + "</hello>";
254.      }
255.
256.  //    http://localhost:8086/GreetingRestService/rest/common/html
257.
258.      @GET
259.      @Path("/html")
260.      @Produces(MediaType.TEXT_HTML)
261.      public String sayHTMLHello() {
262.        return "<html> <body bgcolor='pink'> <h1>
263.                              Welcome to Restful Services </h1> </body> </html>";
264.      }
265.
266.  //    http://localhost:8086/GreetingRestService/rest/common/getJson
267.
268.
269.      @GET
270.        @Path("/getJson")
271.        @Produces(MediaType.APPLICATION_JSON)
272.        public Track getTrackInJSON() {
273.
274.              Track track = new Track();
275.              track.setTitle("Cameraman Ganga tho Rambabu");
276.              track.setSinger("A.R.Rehman");
277.              return track;
278.
279.        }
280.
281.        @POST
282.        @Path("/postJson")
283.        @Consumes(MediaType.APPLICATION_JSON)
284.        public Response createTrackInJSON(Track track) {
285.
286.              String result = "Track saved : " + track;
287.              return Response.status(201).entity(result).build();
288.
289.        }
290.  }
```

Customer.java

```
1. package com.ysreddy.resful.service;
2.
3. import javax.xml.bind.annotation.XmlAttribute;
4. import javax.xml.bind.annotation.XmlElement;
5. import javax.xml.bind.annotation.XmlRootElement;
6.
7. @XmlRootElement(name = "customer")
8. public class Customer {
9.
10.         String name;
11.         int pin;
12.
13.         @XmlElement
14.         public String getName() {
15.               return name;
```

```
16.         }
17.
18.         public void setName(String name) {
19.               this.name = name;
20.         }
21.
22.         @XmlAttribute
23.         public int getPin() {
24.               return pin;
25.         }
26.
27.         public void setPin(int pin) {
28.               this.pin = pin;
29.         }
30.
31.   }
```

Track.java

```
1. package com.ysreddy.resful.service;
2.
3. public class Track {
4.
5.    String title;
6.    String singer;
7.
8.    public String getTitle() {
9.          return title;
10.         }
11.
12.         public void setTitle(String title) {
13.               this.title = title;
14.         }
15.
16.         public String getSinger() {
17.               return singer;
18.         }
19.
20.         public void setSinger(String singer) {
21.               this.singer = singer;
22.         }
23.
24.         @Override
25.         public String toString() {
26.               return "Track [title=" + title + ", singer=" + singer + "]";
27.         }
28.
29.   }
```

Index.html

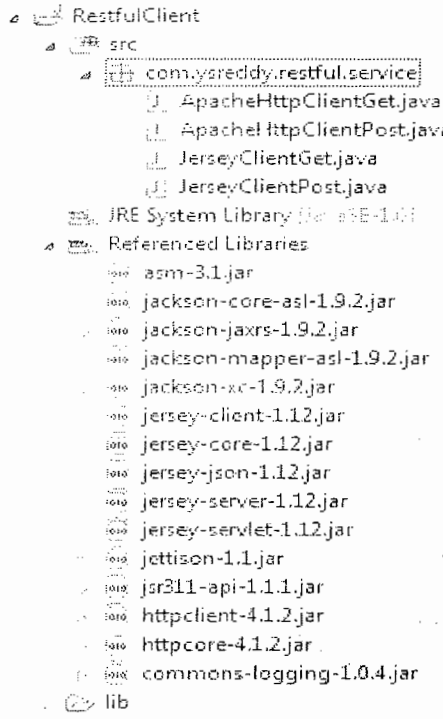
```
1. <html>
2. <body bgcolor="#FFFA00" >
3.    <center>
4.          <h1>Restful Webservcie FormParam Application </h1>
5.          <form action="rest/common/add" method="post" >
```

```
6.                <table>
7.                      <tr>
8.                            <td>Name </td>
9.                            <td> <input type="text" name="name"  /> </td>
10.                           </tr>
11.                           <tr>
12.                                 <td>Age </td>
13.                                 <td> <input type="text" name="age"  /> </td>
14.                           </tr>
15.                           <tr>
16.                                 <td colspan="2" align="center"  >
17.                                  <input type="submit" value="Register"  />
18.                                 </td>
19.                           </tr>
20.                     </table>
21.               </form>
22.         </center>
23.   </body>
24.   </html>
```

Web.xml

```
1. <?xml version="1.0" encoding="UTF-8"?>
2. <web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3. xmlns="http://java.sun.com/xml/ns/javaee"
4. xmlns:web="http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
5. xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
6. http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd" id="WebApp_ID" version="2.5">
7.   <display-name>GreetingRestService</display-name>
8.    <servlet>
9.          <servlet-name>jersey-serlvet</servlet-name>
10.               <servlet-class>
11.                         com.sun.jersey.spi.container.servlet.ServletContainer
12.                   </servlet-class>
13.               <init-param>
14.                    <param-name>com.sun.jersey.config.property.packages</param-name>
15.                    <param-value>com.ysreddy.resful.service</param-value>
16.               </init-param>
17.
18.               <!-- To support json request/responses -->
19.               <init-param>
20.                  <param-name>com.sun.jersey.api.json.POJOMappingFeature</param-name>
21.                    <param-value>true</param-value>
22.               </init-param>
23.               <load-on-startup>1</load-on-startup>
24.         </servlet>
25.
26.         <servlet-mapping>
27.               <servlet-name>jersey-serlvet</servlet-name>
28.               <url-pattern>/rest/*</url-pattern>
29.         </servlet-mapping>
30.
31.   </web-app>
```

Q.) Develop Jersey Client, Http client for Restuful services?



JerseyClientGet.java

```
1. package com.ysreddy.restful.service;
2.
3. import com.sun.jersey.api.client.Client;
4. import com.sun.jersey.api.client.ClientResponse;
5. import com.sun.jersey.api.client.WebResource;
6.
7. public class JerseyClientGet {
8.    public static void main(String[] args) {
9.          try {
10.
11.                    Client client = Client.create();
12.
13.                    WebResource webResource = client.resource(
14.                   "http://localhost:8086/GreetingRestService/rest/common/getJson");
15.
16.                    ClientResponse response = webResource.accept("application/json")
17.                              .get(ClientResponse.class);
18.
19.                    if (response.getStatus() != 200) {
20.                          throw new RuntimeException("Failed : HTTP error code : "
21.                                    + response.getStatus());
22.                    }
23.
24.                    String output = response.getEntity(String.class);
25.
26.                    System.out.println("Output from Server .... \n");
27.                    System.out.println(output);
28.
29.              } catch (Exception e) {
```

```
30.
31.                     e.printStackTrace();
32.
33.                 }
34.
35.         }
36.     }
```

JerseyClientPost.java

```
1. package com.ysreddy.restful.service;
2.
3. import com.sun.jersey.api.client.Client;
4. import com.sun.jersey.api.client.ClientResponse;
5. import com.sun.jersey.api.client.WebResource;
6.
7. public class JerseyClientPost {
8.     public static void main(String[] args) {
9.         try {
10.
11.                     Client client = Client.create();
12.
13.                     WebResource webResource = client
14.                             .resource(
15.                     "http://localhost:8086/GreetingRestService/rest/common/postJson");
16.
17.                 String input="{\"singer\":\"Metallica\",\"title\":\"Fade To Black\"}";
18.
19.                     ClientResponse response = webResource.type("application/json")
20.                             .post(ClientResponse.class, input);
21.
22.                     if (response.getStatus() != 201) {
23.                         throw new RuntimeException("Failed : HTTP error code : "
24.                                 + response.getStatus());
25.                     }
26.
27.                     System.out.println("Output from Server .... \n");
28.                     String output = response.getEntity(String.class);
29.                     System.out.println(output);
30.
31.                 } catch (Exception e) {
32.
33.                     e.printStackTrace();
34.
35.                 }
36.
37.         }
38.     }
```

ApacheHttpClientGet.java

```
1. package com.ysreddy.restful.service;
2.
3. import java.io.BufferedReader;
4. import java.io.IOException;
5. import java.io.InputStreamReader;
6.
```

```
7. import org.apache.http.HttpResponse;
8. import org.apache.http.client.ClientProtocolException;
9. import org.apache.http.client.methods.HttpGet;
10.   import org.apache.http.impl.client.DefaultHttpClient;
11.
12.   public class ApacheHttpClientGet {
13.         public static void main(String[] args) {
14.               try {
15.
16.                     DefaultHttpClient httpClient = new DefaultHttpClient();
17.                     HttpGet getRequest = new HttpGet(
18.                     "http://localhost:8086/GreetingRestService/rest/common/getJson");
19.                     getRequest.addHeader("accept", "application/json");
20.
21.                     HttpResponse response = httpClient.execute(getRequest);
22.
23.                     if (response.getStatusLine().getStatusCode() != 200) {
24.                           throw new RuntimeException("Failed : HTTP error code : "
25.                                       + response.getStatusLine().getStatusCode());
26.                     }
27.
28.                     BufferedReader br = new BufferedReader(new InputStreamReader(
29.                               (response.getEntity().getContent())));
30.
31.                     String output;
32.                     System.out.println("Output from Server .... \n");
33.                     while ((output = br.readLine()) != null) {
34.                           System.out.println(output);
35.                     }
36.
37.                     httpClient.getConnectionManager().shutdown();
38.
39.               } catch (ClientProtocolException e) {
40.
41.                     e.printStackTrace();
42.
43.               } catch (IOException e) {
44.
45.                     e.printStackTrace();
46.               }
47.
48.         }
49.   }
```

ApacheHttpClientPost.java

```
1. package com.ysreddy.restful.service;
2.
3. import java.io.BufferedReader;
4. import java.io.IOException;
5. import java.io.InputStreamReader;
6. import java.net.MalformedURLException;
7.
8. import org.apache.http.HttpResponse;
9. import org.apache.http.client.methods.HttpPost;
10.   import org.apache.http.entity.StringEntity;
```

```
import org.apache.http.impl.client.DefaultHttpClient;

public class ApacheHttpClientPost {
      public static void main(String[] args) {

            try {

                  DefaultHttpClient httpClient = new DefaultHttpClient();
                  HttpPost postRequest = new HttpPost(
                 "http://localhost:8086/GreetingRestService/rest/common/postJson");

                  StringEntity input = new StringEntity(
                              "{\"qty\":100,\"name\":\"iPad 4\"}");
                  input.setContentType("application/json");
                  postRequest.setEntity(input);

                  HttpResponse response = httpClient.execute(postRequest);

                  if (response.getStatusLine().getStatusCode() != 201) {
                        throw new RuntimeException("Failed : HTTP error code : "
                                    + response.getStatusLine().getStatusCode());
                  }

                  BufferedReader br = new BufferedReader(new InputStreamReader(
                        (response.getEntity().getContent())));

                  String output;
                  System.out.println("Output from Server .... \n");
                  while ((output = br.readLine()) != null) {
                        System.out.println(output);
                  }

                  httpClient.getConnectionManager().shutdown();

            } catch (MalformedURLException e) {

                  e.printStackTrace();

            } catch (IOException e) {

                  e.printStackTrace();

            }

      }
}
```