Web Services

realspeed

JAX-RPC

Prerequisite for JAX-RPC

Types of Web Services

➔There are two types of web services

- ⇨ **JAX-RPC (BP 1.0)**
- ⇨ **JAX-WS (BP 1.1)**

➔While developing a Web service, First of we need to choose a type. After choosing the type, we need to select an appropriate implementation and various other aspects.

Web Service Development parts

➔Always a Web Service development involves two parts. Provider and Consumer.

Provider

➔ Provider is the actual Web service, who will handle the incoming requests from the consumer.

Consumer

➔ The Consumer is the Client Program which we develop to access the Provider.

➔JAX-RPC and JAX-WS API's has provided classes and interfaces to develop Consumer as well as Provider programs.

Ways of developing a Web Service

There are two approaches of developing a Web service

➔ Contract First (Top Down Approach)
➔ Contract Last (Bottom up Approach)

Contract First (Top Down Approach)

➔In a Contract First approach, the development of the provider will starts with WSDL.
➔From which the necessary Java classes required to build the service would be generated.
➔As the development starts from WSDL, hence it is called Contract First approach.

Contract Last approach (Bottom up Approach)

➔ In a Contract Last approach, the developer first starts developing the service by writing the Java code.
➔Once done then he will generates the WSDL contract to expose it as a Web Service.
➔As the contract is being generated at the end after the development is done, this is called Contract Last approach.

WSDL (Web Services Description Language)

➔In Web Services, WSDL acts as a contract between Consumer and Provider. If consumer wants to know the information about the provider, he needs to see the WSDL document.

EndPoint

What is end point?

➔End Point is the final destination to whom consumer sends the request to get the business services or the functionalities. From consumer point of view Provider acts as an End Point.
➔When ever consumer sends **through HTTP request** provider should not directly receive the request because of different different protocols (**FTP, SMTP**) provider will get tightly coupled.
➔So there is a HTTP Processor who acts as a provider end point to receiving the request and processing the request.

➔ In java Servlet and EJB are capable of handling incoming HTTP Requests.
➔There are two endpoint based web services are there.

Servlet Endpoint

➔Servlet can receive the request and can forward for further processing.
➔If we develop our end point based on servlet then this is called servlet end point based provider.

EJB Endpoint

➔ If we develop our end point based on EJB then this is called EJB end point based provider.
➔ Most of the people prefer to use Servlet endpoint rather than EJB endpoint

FAQ:-
Which type of provider are you working on?

➔WSI BP 1.0
➔**JAX-RPC API**
- ➔JAX-RPC SI implementation (OR)
- ➔Apache Access implementation (OR)
- ➔Oracle Weblogic Webservices implementation (OR)
- ➔IBM Websphere Webservices implementation (OR)

OR

➔WSI BP 1.1
➔**JAX-WS API**
- ➔JAX-WS RI implementation
- ➔Apache Access2 implementation
- ➔Oracle Weblogic Webservices implementation
- ➔IBM Websphere Webservices implementation

Why People prefer to use Servlet rather than EJB?

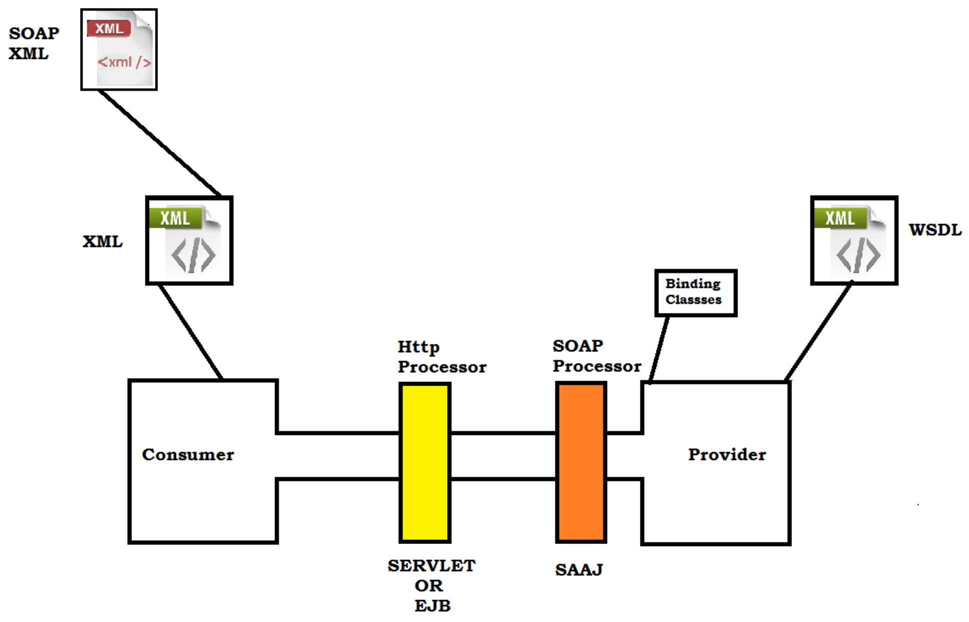
➔As EJB seems to be heavy component and deployment is very costly.

➔

➔And for the sake of developing Web services, the application needs to be hosted on an EJB container.

When we should go for EJB EndPoint?

1. If Provider wants to expose business functionalities to external partner as well as internal like other application that is part of same organization.
2. If both the application of same organization developed in JAVA we don't need interoperability.
3. So both the application from same organization if we are accessing the web services then performance problem will occur. Because they were exchanging the data over the SOAP and XML parsing is costly (Ex: DOM (Tree)).
4. They can directly can access the EJB (it is called as java native calls) so in this case we should go for EJB End-Point.



H E A D E R

POST http://localhost:8080/BookStoreWeb/bookStore HTTP/1.1
Host: localhost
Content-Type: text/xml; charset=utf-8
Content-Length: length
SOAPAction:"http://localhost:8080/BookStoreWeb/bookStore#getBookPrice"

P A Y L O A D

```
<?xml version="1.0"?>
<soap:Envelope xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
          soap:encodingStyle="http://www.w3.org/2001/12/soapencoding">
     <soap:Header/>
     <soap:Body>
          <getBookPrice>
          <isbn xsi:type=”xs:string>ISBN1001</isbn>
          </getBookPrice>
     </soapy:Body>
</soap:Envelope>
```

HTTP REQUEST FORM

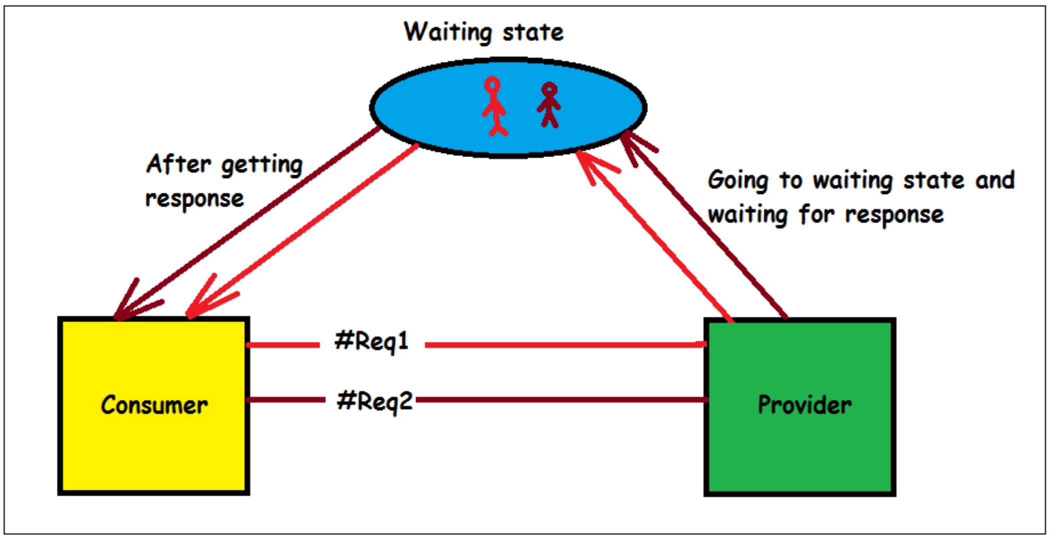
Message Exchange Patterns

➔ The process of exchanging or communicating message or data between the consumer and provider is called MEP(Message Exchange Pattern)
➔There are three ways of message exchange between consumer and provider

- ⇨ **Synchronous request/reply (**Blocked**)**
- ⇨ **Asynchronous request/reply** (Delayed Response)
- ⇨ **One way Invoke** (Fire and forget)

Synchronous request/reply

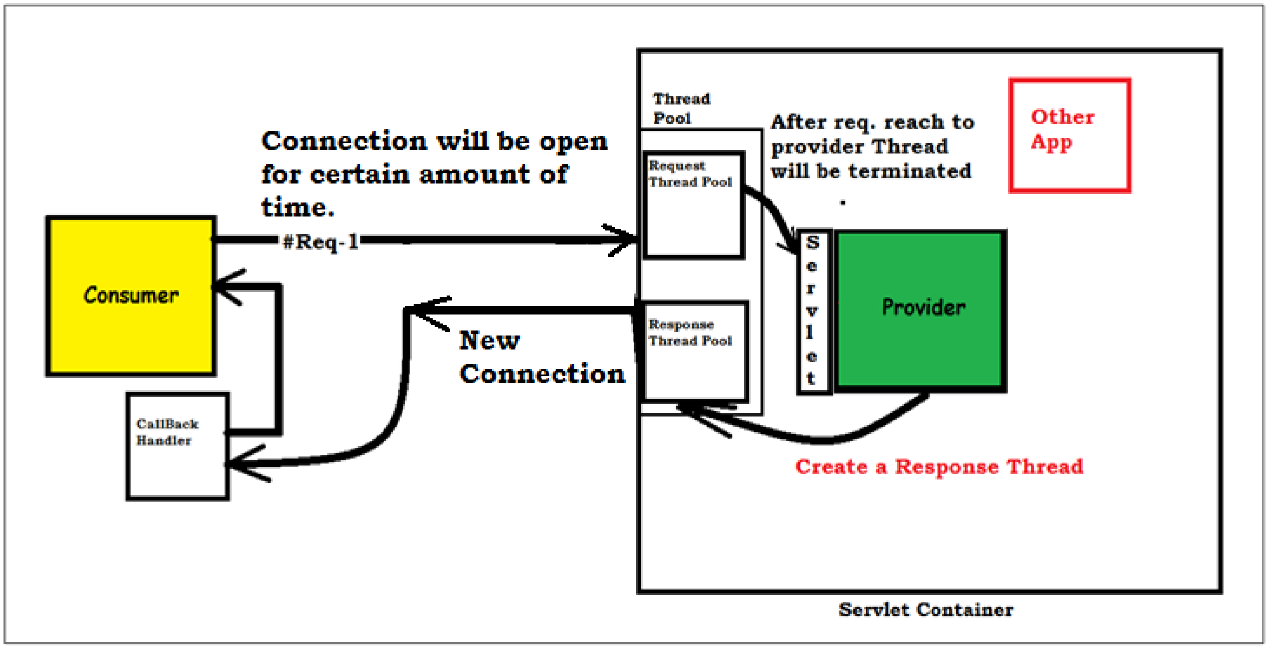
➔In this pattern, the consumer sends the request to the provider through a connecting network and waits till the provider finishes the processing the request and send back the response to the consumer.
➔Until the consumer will not get back the response from the provider the connecting medium will blocked, that's why Synchronous request/reply also called blocked request/response.



Synchronous Request/Reply

Asynchronous request/reply or Delayed Response

➔In this pattern, the consumer sends the request to the provider through a connecting network and consumer not waits for the response back of provider.
➔Consumer will not block the communication network. After the consumer sending the request disconnect the HTTP connection
➔The provider upon receiving the request will hold it and process it after certain amount of time.
➔The provider will receive the request hold it and process it after some time when the response is ready by creating a new connection back to the Consumer and dispatches the response.
➔As the response is not coming immediately and it send after some time so it is called delayed response.



Scalable Asynchronous Request/Reply

Scalable request/response:

➔Consumer send the request to provider over a connection network and the request is first going to servlet container.

➔Inside servlet container **Request Thread Pool** and **Response Thread Pool** are there.

➔When the request is coming to **Request ThreadPool** one of the Thread send the request to Provider.

➔Provider takes the request and process the request and **send back the response to Consumer with the help of ResponseThreadPool.**

➔**Thread has not any return type(because run() return type is void) so this Response Thread pool is not give response to consumer.**

➔To overcome this problem "**Callback handler**" came into the picture.

➔So the Response ThreadPool communicates with consumer by the help of **Call Back Handler.**

➔But in scalable approach the connection is not closed immediately.

Why should we go for Asynchronous request/response?

➔In asynchronous request/response when the consumer send the request to provider over the connecting network then the request first goes to the servlet container.
➔From servlet container it goes to the "**Request Thread Pool**" and from the Request Thread Pool the request thread come to the provider where the request is going to process the data.
➔After that Provider prepare the request and send the data to servlet container.
➔Where the servlet container is going to send the response to the "**Response Thread Pool**" from Response Thread Pool the thread dispatch the response to Consumer.
➔Due to the presence of Thread Pool the waiting time for multiple requests is going to less.
➔So Response Time is going to improve and Improve Scalability so we should go for Asynchronous.

When should we go for Asynchronous request/response?

➔If the consumer's request takes "**Long Execution Services**" then we should go for Asynchronous request/response.

Message Oriented Communication:

➔Consumer send the request to provider over a connection network and the request is first going to servlet container.

➔In this approach the http request contain two part head and payload.

➔Where head contain "**Reply To**" and "**Co-related Id**" and body contain Business logic.

➔If one consumer sends multiple responses then by the help of Correlated Id we can know which response is for which request.

➔Inside servlet container **Request Thread Pool** and **Response Thread Pool** are present.

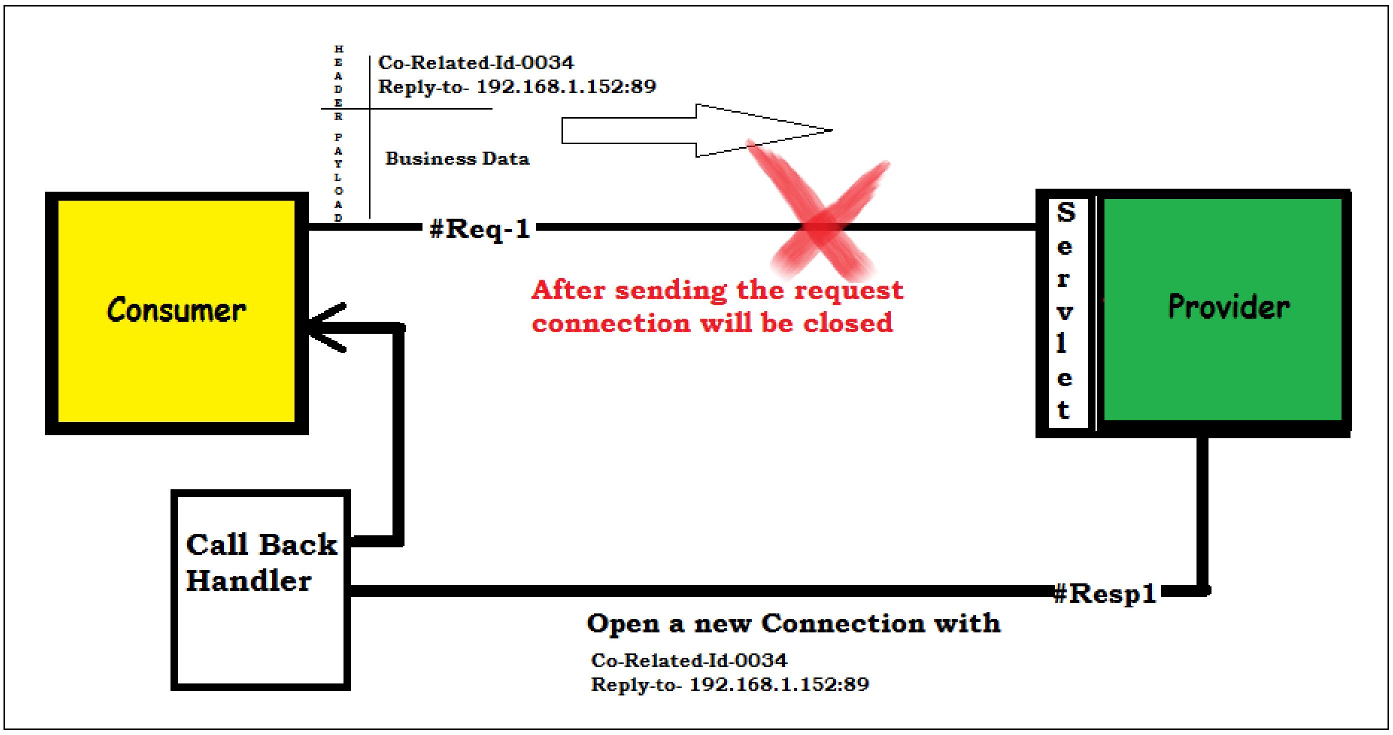
➔When the request is_coming to Request Thread Pool then one of the Thread send the request to Provider.

➔Provider takes the request and placed it in a Queue. and give an acknowledgement to the consumer that he get the request.

➔There may be number of request are present in a queue. so the consumer drop a request and terminate the connection and the provider response back according to the queue number ,

➔Provider takes one after one request and give the response to consumer.

➔Similarly the provider is response back to consumer by header and payload format with the help of callback handler.


HEADER
PAYLOAD
Co-Related-Id-0034
Reply-to- 192.168.1.152:89
Business Data
#Req-1
After sending the request connection will be closed
Consumer
Servlet
Provider
Call Back Handler
#Resp1
Open a new Connection with
Co-Related-Id-0034
Reply-to- 192.168.1.152:89

Message Oriented Asynchronous Request/Reply

One way Invoke or Fire and forget

➔In this pattern the consumer send the request the response to the provider and never wait or expects the response from the provider.
➔This communication is always one directional so it is called one way invoke or Fire and forget.

Why we need & when to use One Way Invoke communication ?

➔When consumer sends a request to the provider, then provider is processing the request but at that time consumer don't want any response back from the provider because the consumer have nothing to do with this response & the responses or the process directly visibility as the outcome in the system & also to improve the band width , then we will go for One Way Invoke .

➔When the provider's response is not going to act as the input for the consumer to act to perform some extra processing then we will goes for Fired & Forget Message pattern.

Example:

➔Insert a record into the data base table is the method that we have written. After calling the method it is not returns any returning value but we can visible the effect in the data base table that itself confirms the process of the method.

➔As the outcome is clearly been known it is generated as an output we not required any return value.

➔When consumer sends the request to the provider it is not mandatory that provider is always successfully process the request. Sometimes provider fails to process the request. At that time also provider will not give any error message response back to the consumer. Because the connection is closed by the consumer after sending the request to the provider.

Message Exchange Format

➔ The way the messages are been exchanges with provider and consumer.
➔This describes about how the consumer will prepare the xml formats that should be exchange to the provider.
➔Messages are communicate from consumer to provider using SOAP over XML.

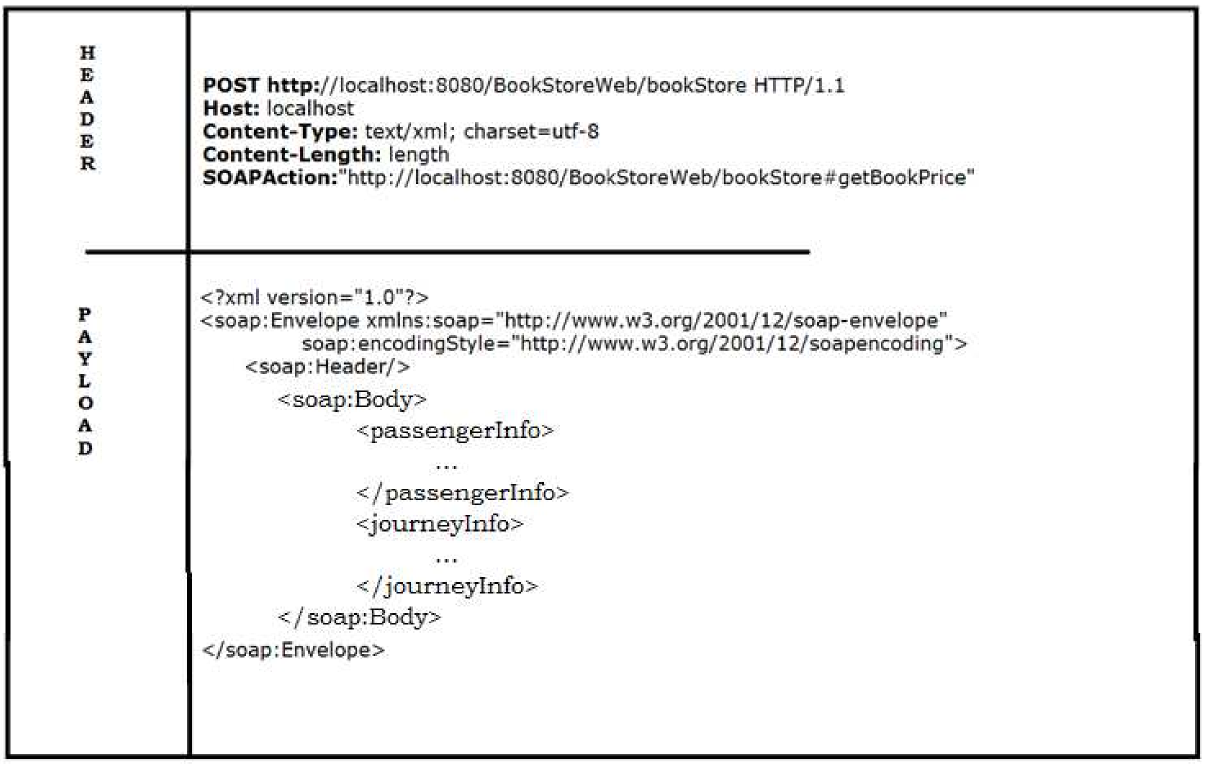
Why we use Message Exchange Format (MEF)?

➔There are two parts in web services.
- ➔Consumer
- ➔Provider

➔ Consumer and provider may developed in java or any other language. It's not required that consumer and provider are in same technology. Then the general and only one way of understand the data by the both is if the data is in XML.

➔ Always the consumer is start communicate with provider. For that consumer create a Http Request and send it to provider via a connection network

➔ The Http request has two part header and payload. Where header contains the header data and payload contains SOAP because SOAP contains the classification information of the consumer.



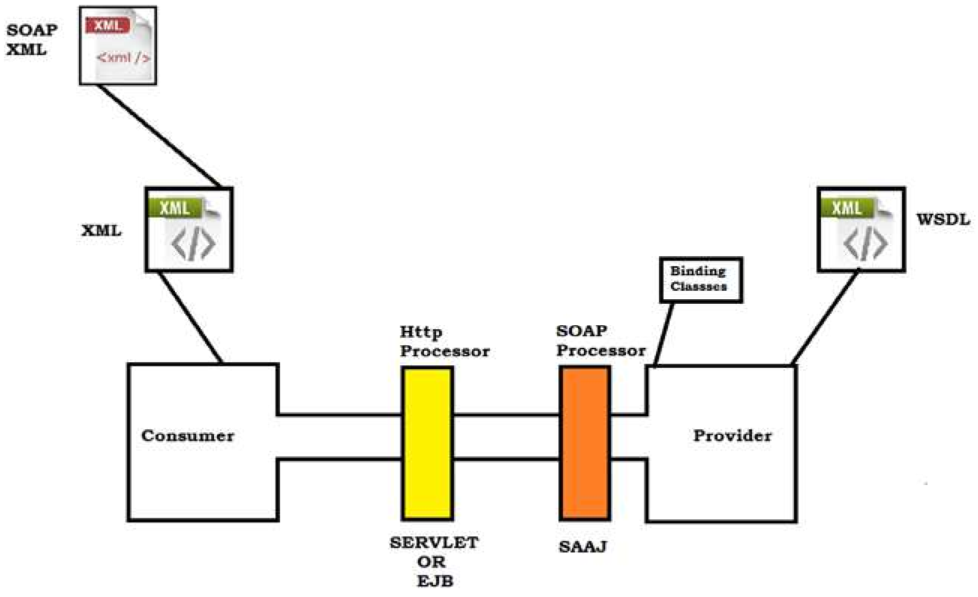
```
POST http://localhost:8080/BookStoreWeb/bookStore HTTP/1.1
Host: localhost
Content-Type: text/xml; charset=utf-8
Content-Length: length
SOAPAction:"http://localhost:8080/BookStoreWeb/bookStore#getBookPrice"
```

```
<?xml version="1.0"?>
<soap:Envelope xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
          soap:encodingStyle="http://www.w3.org/2001/12/soapencoding">
     <soap:Header/>
        <soap:Body>
               <passengerInfo>
                      ...
               </passengerInfo>
               <journeyInfo>
                      ...
               </journeyInfo>
        </soap:Body>
</soap:Envelope>
```

HTTP REQUEST FORM

➔ SOAP have two parts head and body and it start with a <**envelope**>tag and it is also a XML.In header part it contains header data which is optional and the body contains business data which are given as input to provider(element and attribute).

➔ When the consumer send the Http Request to the provider it does not send directly to the provider rather than it goes to the Http Processor which is process the Header of the Http request and extract the payload of the request and cannot understand anything and send it to SOAP Processor.



➔ SOAP Processor extracts the head and goes to the body part and checks the xml is well formed or not (like every xml start with prolog and one and only root element etc...).

➔ A parser can't understand a non parsable xml that's why we need **Message Exchanging format**.

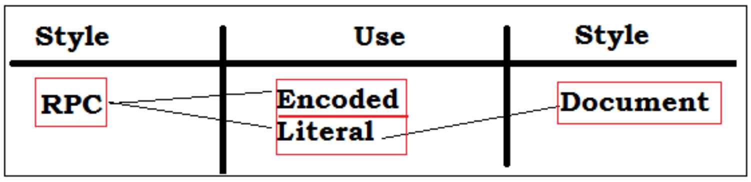
➔ Unless until we not formatted the message provider will not use the data or understand the data so MEF is required.

➔ Provider is mention in which Message Exchanging Format the consumer will send the data to it, for that provider mentioned it in the WSDL and the consumer must follow the WSDL before send the http request.

How many Message Exchanging Format are there?

➔In WSDL the MEF are mentioned in two element format i.e.
- ➔Style
- ➔Use

The possible values for Style are RPC and Document.
The possible values for Use are encoded and literal.



With the above four combinations as

- ➔**rpc-encoded**
- ➔**rpc-literal**
- ➔**document-encoded**
- ➔**document-literal**

Out of which document-encoded is not supported by any web services specification.

Which Web services we are developing?

WS-I Specification	API	Implementation	Approach	End Point	Message Exchange Patterns	Message Exchange Format
1. WS-I BP 1.0	JAX-RPC API	1. JAX-RPC SI Implementation	1. Contract First	1. Servlet End Point	1. Synchronous Request/Reply (Blocked)	1. RPC-Encoded
		2. Apache AXIS	2. Contract Last	2.EJB End Point	2. One way Invoke (Fire & Forget)	2. RPC-Literal
		3. Oracle Web logic Web Service				3. Document-Literal
		4. IBM Web Sphere Web Services				

WS-I Specification	API	Implementation	Approach	End Point	Message Exchange Patterns	Message Exchange Format
1. WS-I BP 1.1	JAX-WS-API	1. JAX-WS-RI	1. Contract First	1. Servlet End Point	1. Synchronous Request/Reply (Blocked)	Document-Literal
		2. Metro	2. Contract Last	2.EJB End Point	2. Asynchronous Request/Reply (Delayed)	
		3. Apache AXIS2			3. One way Invoke (Fire & Forget)	
		4. Apache CFX				
		5. Oracle Web logic Web Services				
		6. IBM Web Sphere Web Services				

JAX-RPC SI IMPLIMENTATION

➔By default any JAX-RPC API based web service uses the default message exchange pattern as RPC-Encoded and implementation is **JAX-RPC SI Implementation**.
➔Before developing a consumer, first they should be a provider to access it. So, let us start first with Provider Development. As you know there are multiple things you need to consider while building provider, we are developing here

WS-I BP1.0 Document Spécification

JAX-RPC API

JAX-RPC SI Implementation

Using Contract Last Approach

Using Servlet Endpoint

Using Sync req/repl (MEP)

Using RPC-Encoded (MEF)

Service Specific Web services.

Here always the service development would be started with Java and at the end, WSDL would generated, as the WSDL is being generated at the end, it is called contract last approach

Step-1(Create SEI Interface)

Service Endpoint (SEI) Interface

➔It describes the entire information about the Provider (service) so it is called Service Endpoint Interface
➔Always from an architecture point of view, the contract between the consumer and the provider is WSDL.
➔But From Java point of view, the contract between consumer and the provider is Interface. It acts as a contract between Consumer and Provider.
➔All the methods declared in the SEI interface are exposed as Web Service methods.
➔There are certain rules you need to follow while writing SEI interface in JAX-RPC API.

What are the rule we have to follow while we are write service end point interface(SEI)?

➔ SEI interface must ensure extend from java.rmi.Remote interface.
➔ Declare only those methods which you want to expose to the web services method.
➔Remote interface is a marker interface. The methods we declare in the Web services has follows some rules

- ➔Method must should have public method.
- ➔ Method should not be final or static.
- ➔The name of the method can be any thing
- ➔The parameter and the return type should be Serializable.
- ➔And all the methods must ensure **declared to throws RemoteException**.

Q.Why the SEI interface should extends from Remote interface?

➔ **To differentiate a normal interface to a remotely accessible interface, it has to extend from Remote interface**
➔Only the methods declared in SEI interface are exposed as Web Service methods or **remote accessible methods that's why SEI interface extends from Remote Interface.**

Q.Why the method should declare to throws RemoteException ?

➔In distributed technology two different programs running in two different machine/ environment over the network.
Example: consumer sends book's ISBN number but sometime provider fails to process the request and report's an error.
➔So Provider always not able to succefully process the request, sometimes failed to process the request.
➔Provider should not send/give any random response to consumer that might put effect on the core business of system of the consumer. So Provider must have to throws a RemoteException.

Why we only throws RemoteException?

➔There in some case where due to the mistake of consumer, consumer is providing wrong data or input so the application may raise RE:ClassNotFound or due to giving wrong business logic application may raise RE:SQLException or some exception are occurs.

➔ In order to differentiate between local exceptions and Web service exceptions, my SEI interface Web service method should always throw RemoteException.

➔To distinguish **between the exception reported** by the **local machine or system** (Consumer) or **remote machine/system (Provider)** we need to throws RemoteException (**checked Exception**) by the provider.

➔Even your method is declared to throw RemoteException, the code inside your method may not be throwing exception at all.

If the method may or may not throw Exception why should I declare it to throw in Provider side?

➔Your code may not be throwing the exception, but while client sending a request to the provider.
➔There could be **some network failures or due to some wrong input**, consumer application **may fails at runtime so to avoid such types of failures**, **consumer has to handle Exceptions so** at provider side all the methods are throwing RemoteException to the User.
➔So by declaring RemoteException provider tells always preparing the consumer to handle RemoteException.

Q Why RemoteException are designed to be checked Exception?

➔If the exception that arise and the consumer has not notified. Then the consumer will not understand the reason behind the exception/failure.
➔It's so important to know the reason behind the failure because **the failure not only occurs because of** provider sometimes it occurs due to the consumer and the network problem.

➔So to identify the cause of failure and to solve the problem and handle the exception the RemoteException must be checked exception.

Consumer Method

```
public class BookInfo {
        public void getBookPrice(int isbn){
                try{
                        BookInfo bookInfo = //call the Provider methods and get BookInfo;
                        //perform something using bookInfo
                }catch(RemoteException e){
                }catch(NullPointerException e){
                }
        }
}
```

How do I need to find whether the exception was caused due to Web service call or the local code has thrown the error?

➔If the type of Exception being caught is RemoteException, it indicates a Remote program has caused the error.
➔Otherwise would be considering as local code has raised the exception.

```
public interface IBookInfo extends Remote{
        public void getPrice(int isbn)throws RemoteException;
}
```

STEP-2(Create Implementation class)

Write an implementation class
➔It must implements from SEI interface.
➔Override all the method from SEI interface.
➔Inside the implementation class we have to write complex business logic (because provider always contain complex business logic). Our implementation class may occur any exception or may not.
➔The SEI interface throws RemoteException so it's implementation class which by default goes for an exception.
➔These methods must be follow all the rules of SEI interface but it may or may not throws RemoteException always.
➔ The implementation class methods will contain business logic to handle and process the incoming requests, but the logic you wrote here may or may not raise any exceptions. In a case where your logic is causing any exceptions, you don’t need to declare your method to throw RemoteException.
➔ But if your business logic is raising a NullPointerException or SqlException, the class should not throw NullPointerException or SqlException rather those exceptions has to be caught in try/catch block and should wrap those exceptions into RemoteException and should throw to the Consumer.

```
class BookInfoImpl implements IBookInfo{
        @Override
        public float getPrice(int isbn) throws RemoteException {
                float price=0.00f;
                try {
                        //some DB operation
                } catch ( SQLException | NullPointerException  e) {
                        throw new RemoteException("...");
                }
                return price;
        }
}
```

STEP 3 :(Create Binding classes)

➔What is binding class? And why we are use binding class? And how we create binding class in Contract last approach?

➔We need binding class for converting XML represented data to object represent format and send to provider to process the object .
➔After process the object binding class we convert the object represented format to XML represented format using binding classes.
➔For creating binding class we need structural representation of XML i.e. XSD
➔Without Binding class the provider cannot understand the consumer's request, so Binding class is required.
➔In contract last approach we don't have XSD to represent the structure of XML so first we need structure of the XML to create the binding class.
➔We can create binding class by looking at method, Return type and its parameter type.
➔Let's see how one can create XML by looking at method

```
public interface BookTicketService extends Remote{
        public Ticket bookTicket(journeyInfo jInfo, PassengerInfo pInfo);
}
```

Request XML

```
<?xml version="1.0" encoding="UTF-8"?>
<bookTicket>
    <jInfo>
        <source></source>
        <destination></destination>
        <journeyDate></journeyDate>
    </jInfo>
    <pInfo>
        <uniqueId></uniqueId>
        <name></name>
        <age></age>
        <mobileno></mobileno>
        <gender></gender>
    </pInfo>
</bookTicket>
```

Response XML

```
<?xml version="1.0" encoding="UTF-8"?>
<bookTicket>
    <result>
        <ticketNo></ticketNo>
        <amount></amount>
        <jInfo>
            <source></source>
            <destination></destination>
            <journeyDate></journeyDate>
        </jInfo>
        <pInfo>
            <uniqueId></uniqueId>
            <name></name>
            <age></age>
            <mobileno></mobileno>
            <gender></gender>
        </pInfo>
    </result>
</bookTicket>
```

➔In response XML the sub-root element is result because method return type there is no name only the type is available

➔But when we are creating a binding classes by looking at its a tedious job rather JAX-RPC API has provides some tools to generate binding classes.

Generate Binding classes – After developing the SEI interface and Implementation classes we need to run a tool called wscompile to generated binding classes.

What tools are available?

➔The wscompile tool is shipped as part of JWSDP.
➔For developing a JAX-RPC API RI based Web service we need some set of tools/compilers to generate some classes similar to we generate in JAX-B, but where are these tools under?
➔These tools are available to you when you install JWSDP. Basically we will use two tools for JAX-RPC API RI based web service development.

➔Wscompile – Used for generating binding classes from SEI interface or WSDL
➔Wsdeploy – Used for generating service related configurations which are Required to deploy your service.

Both these tools are shipped as part of JWSDP when you install and are available under the below folder.

```
C:>Sun\JWSDP-2.0
      | - jaxrpc
            | - bin
                 | - wscompile.bat
                 | - wsdeploy.bat
```

Why to run wscompile?

➔Now let us try to understand why I need to generate binding classes?
➔As we know the Consumer will send's the request to the Provider requesting for some data or processing.
➔Here sending the request to the Provider is nothing but invoking a method on the Provider for e.g. in the above code he will call getBookPrice method to get the price of the book.
➔While calling this method, the consumer needs to send data required for executing the method.
➔Does the consumer need to send the Java type data?
➔He shouldn't rather send XML equivalent of it for example if it is rpc-encoded, the request XML will look as shown below.

```
<getBookPrice>
     <isbn xsi:type="xs:string">ISBN1001</isbn>
</getBookPrice>
```

➔ "RPC" stands for remote procedural call, so the request XML the consumer is sending would exactly represent like a method call with root element equal to method name and parameters of the method as sub-elements.

This XML would be placed inside SOAP XML as shown below.

```
<?xml version="1.0"?>
<soap:Envelope xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
    <soap:Header/>
    <soap:Body>
        <getBookPrice>
            <isbn xsi:type="xs:string>ISBN1001</isbn>
        </getBookPrice>
    </soapy:Body>
</soap:Envelope>
```

And the above SOAP XML will be placed inside the HTTP Message and sends a HttpRequest to Provider.

Header

```
POST http://localhost:8080/BookStoreWeb/bookStore HTTP/1.1
Host: localhost
Content-Type: text/xml; charset=utf-8
Content-Length: length
SOAPAction:
"http://localhost:8080/BookStoreWeb/bookStore#getBookPrice"
```

Payload

```
<?xml version="1.0"?>
<soap:Envelope
        xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
        soap:encodingStyle="http://www.w3.org/2001/12/soap-
encoding">
        <soap:Header/>
        <soap:Body>
                <getBookPrice>
                        <isbn xsi:type="xs:string>ISBN1001</isbn>
                </getBookPrice>
        </soapy:Body>
</soap:Envelope>
```

Now the above Http Request message would be sent to the Provider. On the Provider side, they should be some person acting as a listener who can receive the above HTTP Request. This listener at the Provider end typically would be a Servlet as he is the best person to process HTTP requests.

Upon Servlet receiving the request, as it can understand HTTP message, it would crack the message and process HTTP Headers. After processing the HTTP header, now it tries to access the PAYLOAD, now the payload portion of it contains SOAP XML which cannot be understood by Servlet, so it simply passes it down to SOAP Processors.

SOAP Processors are the components who know how to parse a SOAP XML, SOAP Processors will process the SOAP Headers if any and then extracts the Body of the Soap Message. The body of the SOAP XML is business XML. Now the SOAP Processors don’t know how to process business XML, so they delegate this XML to Binding classes.

Binding classes are the classes who know how to convert XML to Java Object. Now these classes convert the XML into Java Object from which, runtime takes the data and passes them as parameters in calling the Implementation class methods.

Along with config file we need some more switches while running the tool i.e.

➔”**-d src**”: location where the source and class files will be generate.

➔"**-gen:server** "**:**The binding class which are generated by wscompile tool are not only required by consumer ,but also required by provider to exchange data from xml to object and object to xml.but while we run the tool we have to specify whether we want to run it server side or client side. Here -gen:server indicates **Provider side classes.**

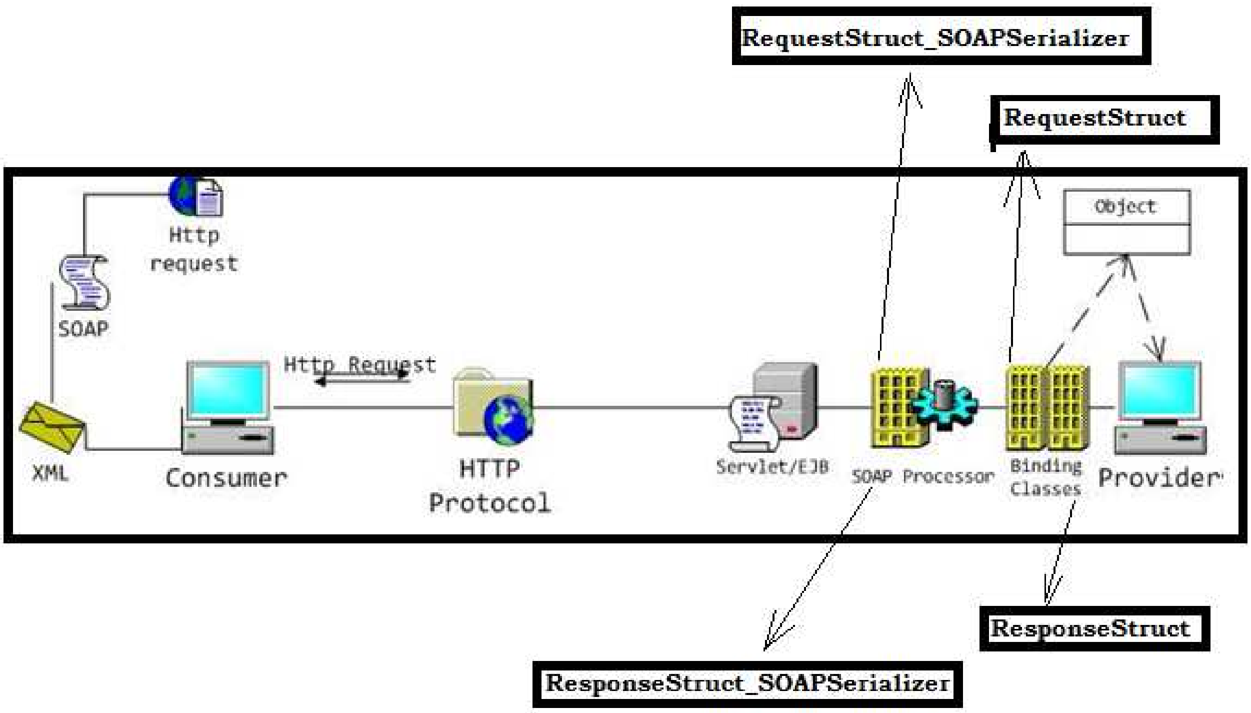
➔"**-cp build\classes** "**:**wscompile take the SEI interface and genetate binding classes.so inorder to read the methed signatures in the SEI interface wscompile take the location/specify the location of class files using "-cp"

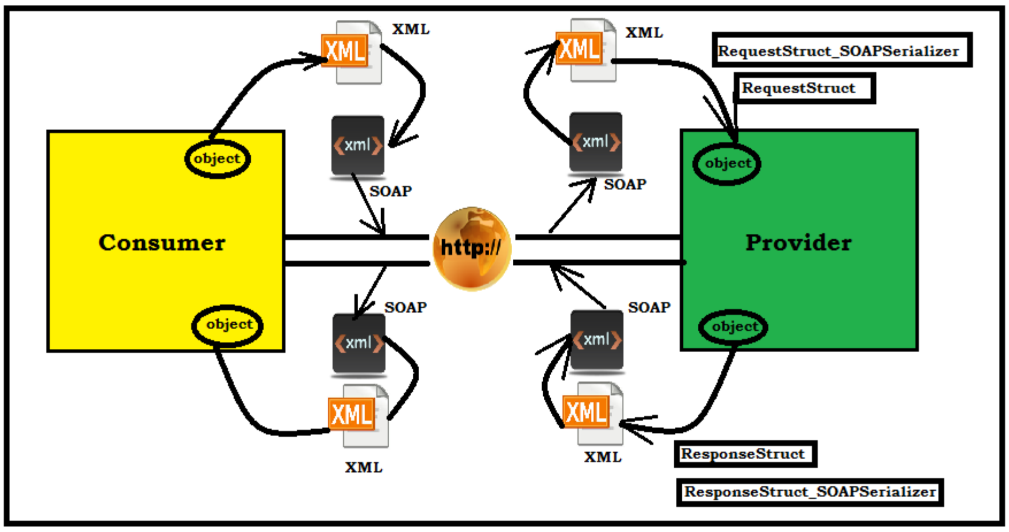
➔"**-keep** "**:wscompile** tool takes the SEI interface and the message encoding format and generating the binding classes.After that it will **compiles the binding classes and delete** the generated source files.**If we want to keep the sourse files** which are generated we need to use "**-keep**" .

➔”**-verbose**”:It will display in console ,what are the files wscompiler generates.

➔"**-model bsmodel-rpc-enc.xml.gz** "**: model** is a JAX RPC SI implementation proprietary file. This file will not be present across all the implimentations, so it not a standard file like wsdl.
Inside the model all the internal data structure information related to methods and how they are parsed of the provider side are describe inside model file. wscompile generate the model file. It is mandatory to generate this file. It is a vender specific.

➔” **WebContent\WEB-INF\config.xml**”**:**Config file location





Example:

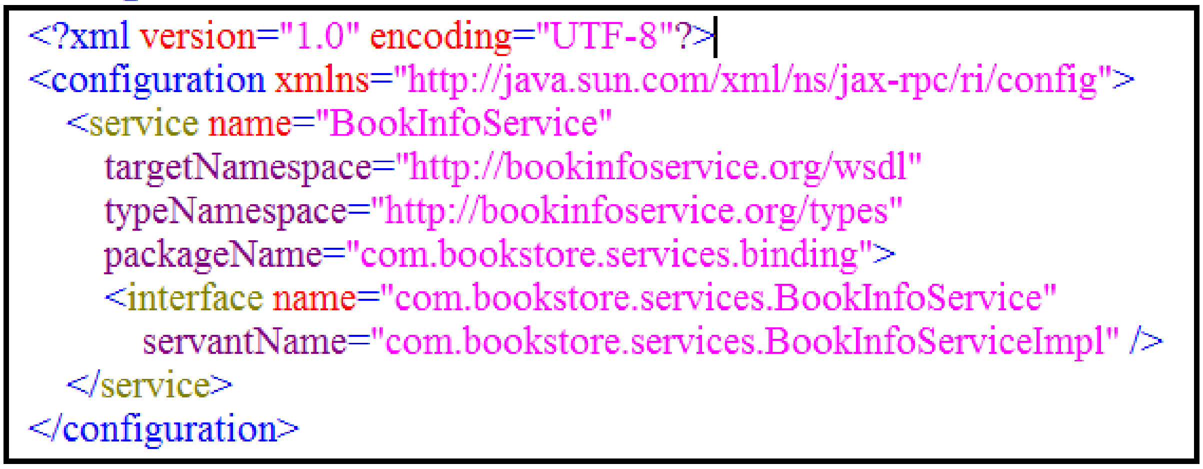
SEI Interface

```
package com.bookstore.services;

import java.rmi.Remote;
import java.rmi.RemoteException;

public interface BookInfoService extends Remote {
    float getBookPrice(String isbn)throws RemoteException;
}
```

Servant (implementation of SEI Interface)

```
package com.bookstore.services;
public class BookInfoServiceImpl implements BookInfoService{
    public float getBookPrice(String isbn) {
        if (isbn.equals("isbn1001")) {
            return 541.32f;
        }else if(isbn.equals("isbn1002")){
            return 884.32f;
        }
        else{
            return 555.32f;
        }
    }
}
```

config.xml



```
<?xml version="1.0" encoding="UTF-8"?>
<configuration xmlns="http://java.sun.com/xml/ns/jax-rpc/ri/config">
  <service name="BookInfoService"
    targetNamespace="http://bookinfoservice.org/wsdl"
    typeNamespace="http://bookinfoservice.org/types"
    packageName="com.bookstore.services.binding">
    <interface name="com.bookstore.services.BookInfoService"
      servantName="com.bookstore.services.BookInfoServiceImpl" />
  </service>
</configuration>
```

How to run the wscompile tool using command?

Set the path (OR) set the environment variable

```
C:\>set path=%path%;C:\Sun\jwsdp-2.0\jaxrpc\bin
```

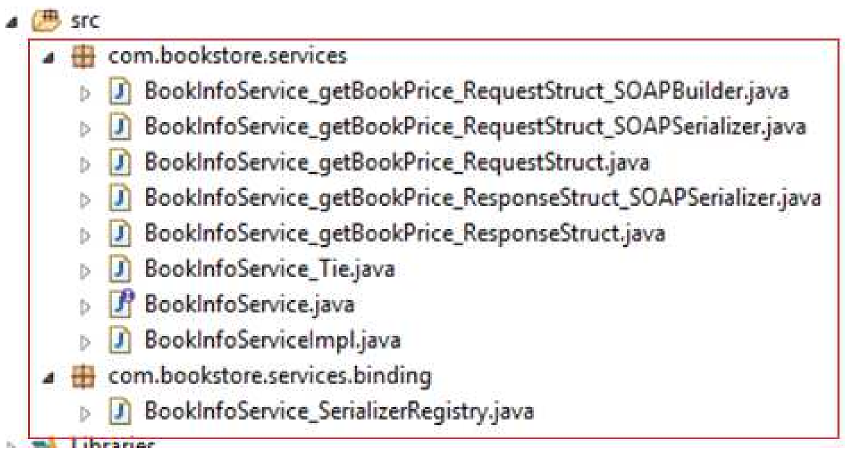
Go to Project Location

```
C:\>cd workspace\JAX-RPC
```

Run wscompile with switches

```
C:\workspace\JAX-RPC>wscompile -d src  -gen:server -cp build\classes -keep -verbose -model bsmodel-rpc-enc.xml.gz WebContent\WEB-INF\config.xml
```

➔With the above command, the wscompile reads the contents of the config.xml and identifies the SEI interface and Implementation class and validates against all the rules in implementing them. Once those are valid, it starts reading the methods of the SEI interface and generates the following artifacts



```
src
  com.bookstore.services
    BookInfoService_getBookPrice_RequestStruct_SOAPBuilder.java
    BookInfoService_getBookPrice_RequestStruct_SOAPSerializer.java
    BookInfoService_getBookPrice_RequestStruct.java
    BookInfoService_getBookPrice_ResponseStruct_SOAPSerializer.java
    BookInfoService_getBookPrice_ResponseStruct.java
    BookInfoService_Tie.java
    BookInfoService.java
    BookInfoServiceImpl.java
  com.bookstore.services.binding
    BookInfoService_SerializerRegistry.java
```

RequestStruct	Representing the structure of requestXml
ResponseStruct	Representing the structure of Response xml
RequestStructSOAPSerializer/Builders	These classes knows how to convert Request XML into RequestStruct class Objects
ResponseStructSOAPSerializer	This knows how to convert ResponseStruct object to response XML
Tie	Who facilitates the processing of Request
Model	Contains the internal data structures Using which we can recreate the Service
WSDL	Describes the information about the Service

If you observe the above, the WSDL has been generated taking the input as Java classes and hence this is called Contract Last approach.

➔After generating the necessary binding classes to expose our class as service, we need to configure the endpoint information of our service in a configuration file called jaxrpc-ri.xml

➔As how we write web.xml to attach for a Servlet an URL to access, for even your Web service class we need to attach an URL to expose it as an service so that Consumers can access the service by using the URL we specified here.

➔Here we configure the complete endpoint information along with attaching it to an URL to access as shown below.

Note: - The file name should be jaxrpc-ri.xml and it is mandatory to use the same file name you should place this file under WEB-INF directory only.

jaxrpc-ri.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<webServices
   xmlns="http://java.sun.com/xml/ns/jax-rpc/ri/dd"
   version="1.0"
   targetNamespaceBase="http://bookinfoservice.org/wsdl"
   typeNamespaceBase="http://bookinfoservice.org/types"
   urlPatternBase="/bookservice">
   <endpoint
      name="BookInfoService"
      displayName="Book Service"
      description="A simple web service"
      wsdl="/WEB-INF/BookInfoService.wsdl"
      interface="com.bookstore.services.BookInfoService"
      implementation="com.bookstore.services.BookInfoServiceImpl"
       model="/WEB-INF/BSModel-rpc-enc.xml.gz">
   </endpoint>
   <endpointMapping
      endpointName="BookInfoService"
      urlPattern="/bookservice"/>
</webServices>
```

➔After writing the jaxrpc-ri.xml file, we need to run wsdeploy tool which generates web service deployment descriptor that carries the information describing the service, which is used for servicing the request from the consumer.

➔While running this tool we need to give the input as war of our application. So first we need to export the project and then need to run the wsdeploy tool by giving this war as input shown below.

How to run the wsdeploy tool using command?

➔Put the WSDL File and Model File inside WEB-INF along with jaxrpc-ri.xml.

➔Set the path (OR) set the environment variable

```
C:\>set path=%path%;C:\Sun\jwsdp-2.0\jaxrpc\bin
```

➔Create a directory under C:\jax-rpc-wars

➔Export your Eclipse Project as WAR file and place it in side **C:\jax-rpc-wars\JAX-RPC.war**

```
C:\jax-rpc-wars>wscompile –o target.war  JAX-RPC.war
```

➔The above command reads the contents of JAX-RPC.war and identifies the web.xml and jaxrpc-ri.xml.

➔It reads the contents of web.xml and updates it with some more additional configuration.

➔Along with this it uses the jaxrpcri. xml as input file and generates web service deployment descriptor jaxrpc-riruntime. xml which contains the configuration related to service we are exposing.

➔These two files will be generated and placed inside the target.war. Now we can either directly deploy the target.war on the Application Server.

➔ Or extract the target.war and copy web.xml and jaxrpc-ri-runtime.xml and can paste our eclipse project inside WEB-INF DIRECTORY and deploy from eclipse as well.

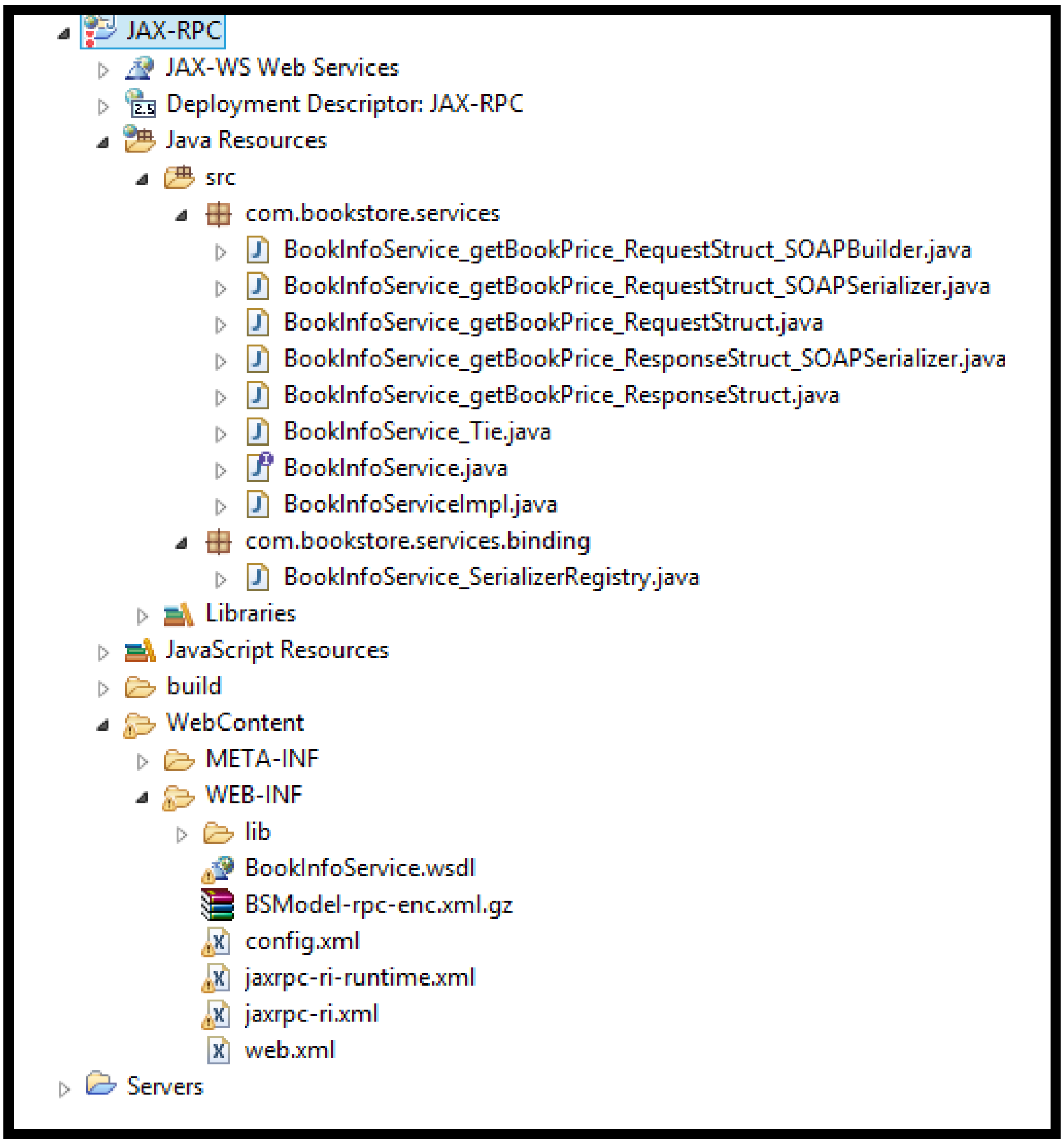
jaxrpc-ri-runtime.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<endpoints xmlns='http://java.sun.com/xml/ns/jax-rpc/ri/runtime' version='1.0'>
 <endpoint
   name='BookInfoService'
   interface='com.bookstore.services.BookInfoService'
   implementation='com.bookstore.services.BookInfoServiceImpl'
   tie='com.bookstore.services.BookInfoService_Tie'
   model='/WEB-INF/BSModel-rpc-enc.xml.gz'
   wsdl='/WEB-INF/BookInfoService.wsdl'
   service='{http://bookinfoservice.org/wsdl}BookInfoService'
   port='{http://bookinfoservice.org/wsdl}BookInfoServicePort'
   urlpattern='/bookservice'/>
</endpoints>
```

➔wsdeploy tool just added tie attribute in jaxrpc-ri-runtime.xml.
➔and configure JAXRPCServlet in web.xml file.
➔because programmer don't worry about what are the classes generated by wscompile, so that sun people has given the wsdeploy tools for generate these files

web.xml

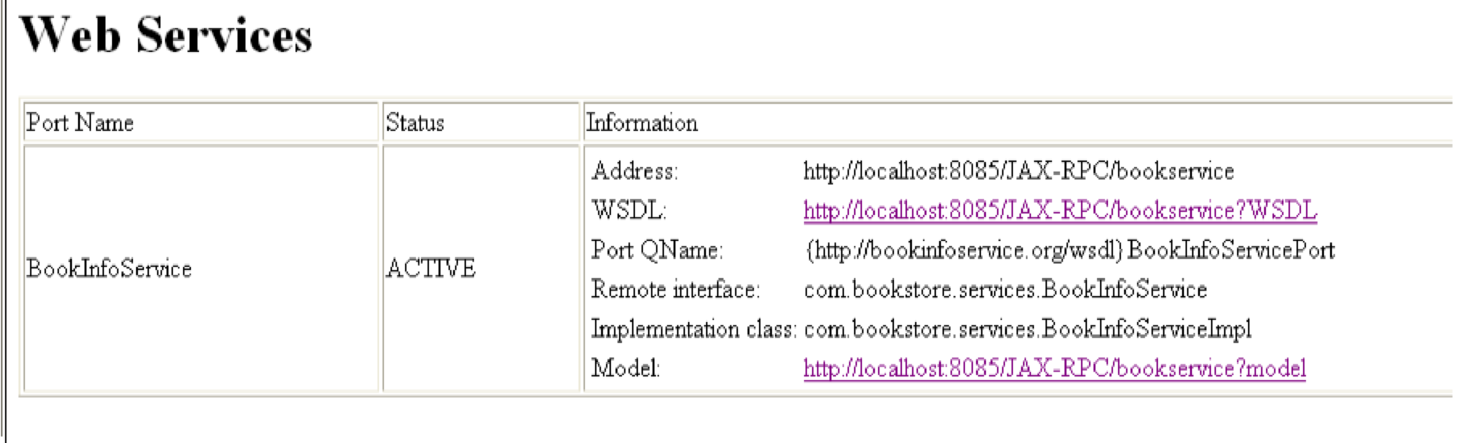
```xml
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://java.sun.com/xml/ns/javaee"
   xmlns:web="http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" id="WebApp_ID" version="2.5"
   xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd">
  <display-name>JAX-RPC</display-name>
  <listener>
      <listener-class>com.sun.xml.rpc.server.http.JAXRPCContextListener</listener-class>
</listener>
<servlet>
      <servlet-name>BookInfoService</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>BookInfoService</servlet-name>
      <url-pattern>/bookservice</url-pattern>
</servlet-mapping>
</web-app>
```



```
◢ JAX-RPC
    ▷ JAX-WS Web Services
    ▷ Deployment Descriptor: JAX-RPC
    ◢ Java Resources
        ◢ src
            ◢ com.bookstore.services
                ▷ BookInfoService_getBookPrice_RequestStruct_SOAPBuilder.java
                ▷ BookInfoService_getBookPrice_RequestStruct_SOAPSerializer.java
                ▷ BookInfoService_getBookPrice_RequestStruct.java
                ▷ BookInfoService_getBookPrice_ResponseStruct_SOAPSerializer.java
                ▷ BookInfoService_getBookPrice_ResponseStruct.java
                ▷ BookInfoService_Tie.java
                ▷ BookInfoService.java
                ▷ BookInfoServiceImpl.java
            ◢ com.bookstore.services.binding
                ▷ BookInfoService_SerializerRegistry.java
        ▷ Libraries
    ▷ JavaScript Resources
    ▷ build
    ◢ WebContent
        ▷ META-INF
        ◢ WEB-INF
            ▷ lib
              BookInfoService.wsdl
              BSModel-rpc-enc.xml.gz
              config.xml
              jaxrpc-ri-runtime.xml
              jaxrpc-ri.xml
              web.xml
◢ Servers
```

➔Deploy the war and start your server and you should be able to access the service.

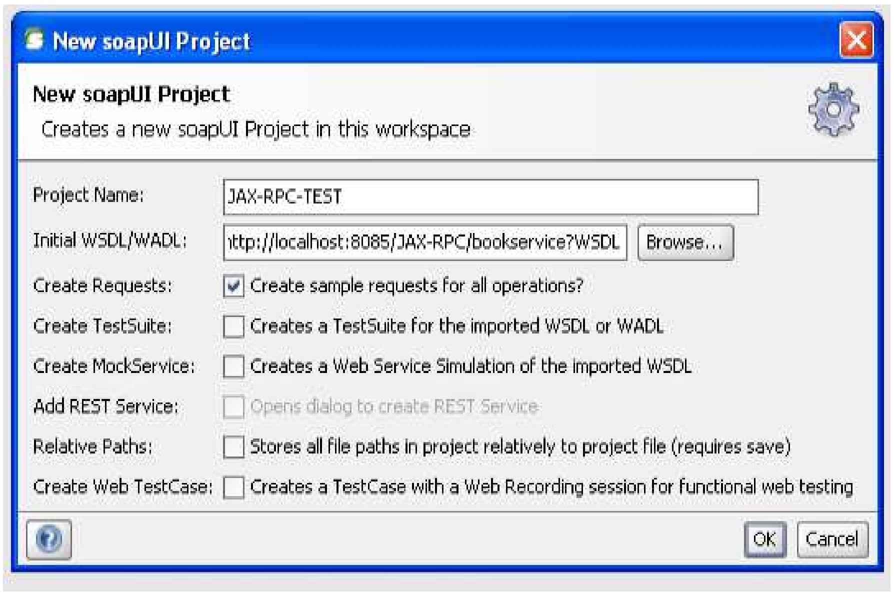
Access http://localhost:8085/JAX-RPC/bookservice



Web Services

Port Name	Status	Information
BookInfoService	ACTIVE	Address: http://localhost:8085/JAX-RPC/bookservice WSDL: http://localhost:8085/JAX-RPC/bookservice?WSDL Port QName: {http://bookinfoservice.org/wsdl}BookInfoServicePort Remote interface: com.bookstore.services.BookInfoService Implementation class: com.bookstore.services.BookInfoServiceImpl Model: http://localhost:8085/JAX-RPC/bookservice?model

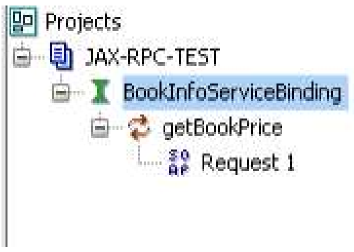
➔To check the request and response we have to use an application **SOAP-UI**
➔**Install SOAP-UI in your system**
➔**Open SOAP-UI and create a Project**
➔ **And give the WSDL file Link to the SOAP-UI**



New soapUI Project
Creates a new soapUI Project in this workspace

Project Name:	JAX-RPC-TEST
Initial WSDL/WADL:	ttp://localhost:8085/JAX-RPC/bookservice?WSDL
Create Requests:	✓ Create sample requests for all operations?
Create TestSuite:	☐ Creates a TestSuite for the imported WSDL or WADL
Create MockService:	☐ Creates a Web Service Simulation of the imported WSDL
Add REST Service:	☐ Opens dialog to create REST Service
Relative Paths:	☐ Stores all file paths in project relatively to project file (requires save)
Create Web TestCase:	☐ Creates a TestCase with a Web Recording session for functional web testing

Then a Project will be created



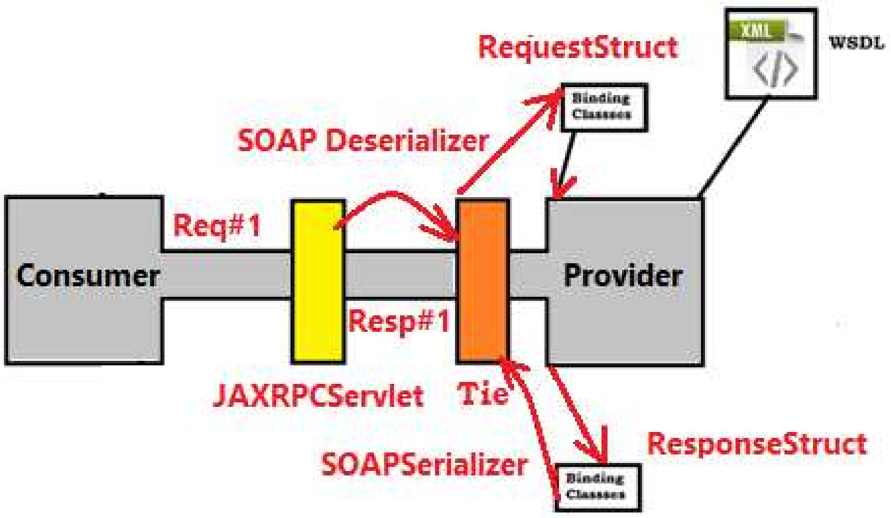
Double click on Request 1

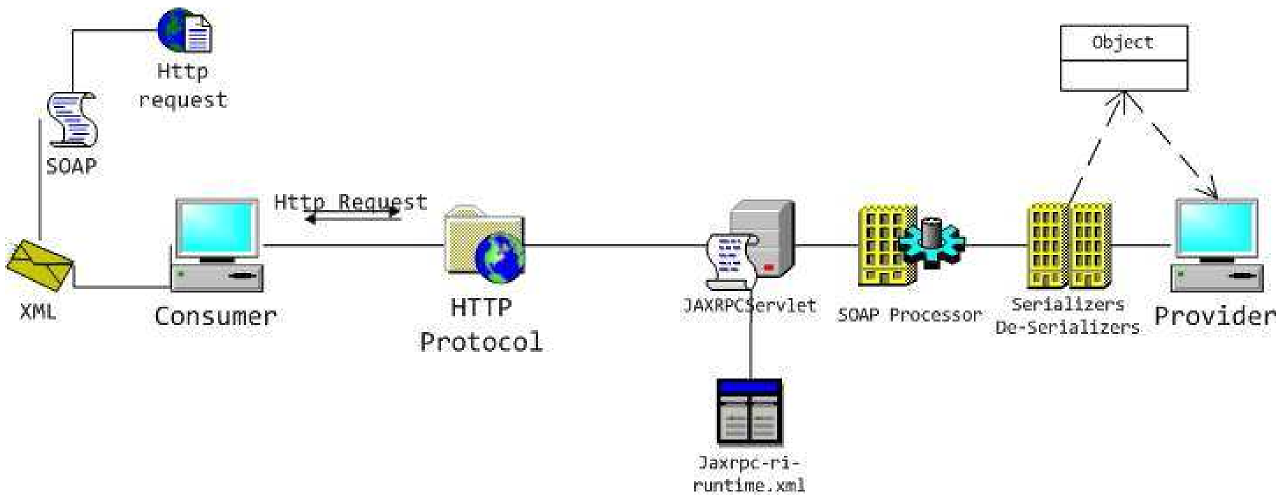
<u>Request Xml Generated by SOAP-UI</u>

```
<soapenv:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xmlns:xsd="http://www.w3.org/2001/XMLSchema"
          xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
          xmlns:wsdl="http://bookinfoservice.org/wsdl">
   <soapenv:Header/>
   <soapenv:Body>
      <wsdl:getBookPrice soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
         <String_1 xsi:type="xsd:string">isbn1001</String_1>
      </wsdl:getBookPrice>
   </soapenv:Body>
</soapenv:Envelope>
```

Response Xml Getting from Provider

```
<env:Envelope env:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
     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://bookinfoservice.org/types">
   <env:Body>
      <ans1:getBookPriceResponse xmlns:ans1="http://bookinfoservice.org/wsdl">
         <result xsi:type="xsd:float">541.32</result>
      </ans1:getBookPriceResponse>
   </env:Body>
</env:Envelope>
```





Internals

RequestStruct

```
package com.bookstore.services;
public class BookInfoService_getBookPrice_RequestStruct {
   protected String String_1;

      //no-arg Constructor
   public BookInfoService_getBookPrice_RequestStruct() {
   }
      //parameterize Constructor
   public BookInfoService_getBookPrice_RequestStruct(String String_1) {
      this.String_1 = String_1;
   }

      //getter() Method
   public String getString_1() {
      return String_1;
   }
      //setter Method
   public void setString_1(String String_1) {
      this.String_1 = String_1;
   }
}
```

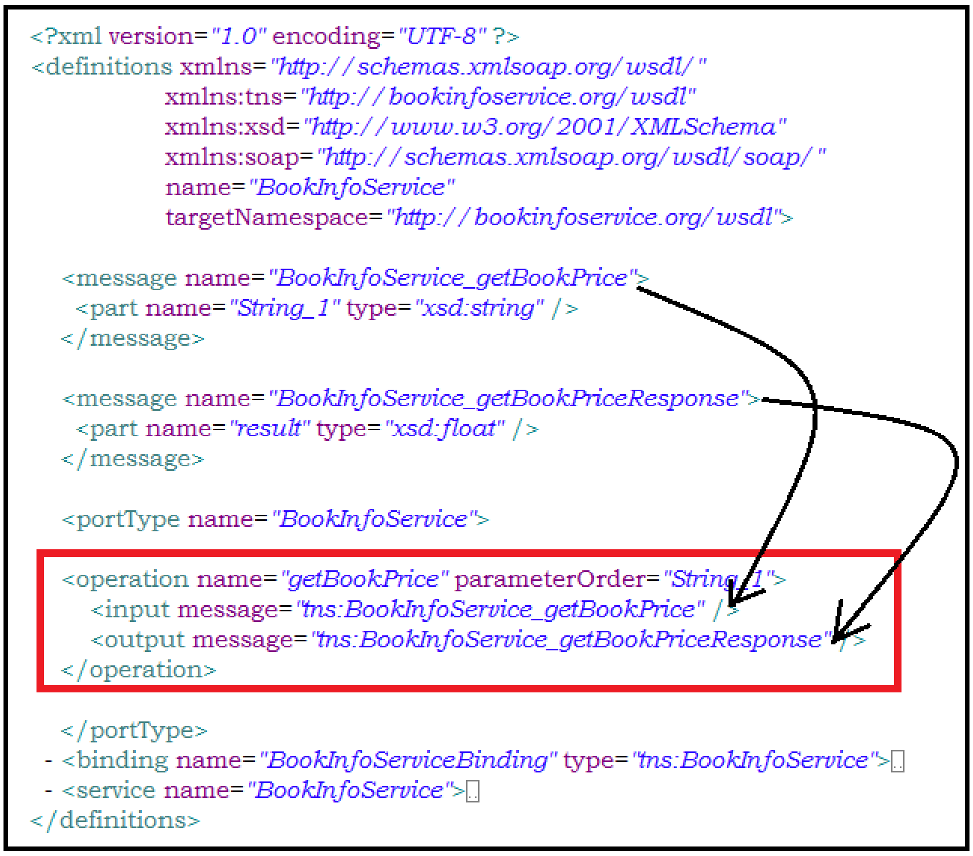
ResponseStruct

```
package com.bookstore.services;
public class BookInfoService_getBookPrice_ResponseStruct {
   protected float result;

      //no-arg Constructor
   public BookInfoService_getBookPrice_ResponseStruct() {
   }

      //Parameterize Constructor
   public BookInfoService_getBookPrice_ResponseStruct(float result) {
      this.result = result;
   }

      //getter () Method
   public float getResult() {
      return result;
   }
      //setter () Method
   public void setResult(float result) {
      this.result = result;
   }
}
```

Tie

```
public class Tie extends TieBase implements SerializerConstants {

    //Constructor
   public Tie() throws Exception {
      super(new SerializerRegistry().getRegistry());
      initialize(internalTypeMappingRegistry);
   }

    //This method does the actual method invocation for operation: getBookPrice
   private void invoke_getBookPrice(StreamingHandlerState state) throws Exception {
                           //Request Processing

      RequestStruct requestStruct = null;
      Object requestStructObj=state.getRequest().getBody().getValue();

      if ( requestStructObj instanceof SOAPDeserializationState) {
         requestStruct = (RequestStruct)((SOAPDeserializationState)requestStructObj).getInstance();
      } else {
        requestStruct = (RequestStruct)requestStructObj ;
      }


      try {              //Response processing
        float result =
               ((BookInfoService) getTarget()).getBookPrice(requestStruct.getString_1());
          ResponseStruct responseStruct =new ResponseStruct();
        SOAPHeaderBlockInfo headerInfo;
        responseStruct.setResult(result);

        SOAPBlockInfo bodyBlock = new SOAPBlockInfo(Response_QNAME);
        bodyBlock.setValue(responseStruct);
        bodyBlock.setSerializer(ResponseStruct_SOAPSerializer);
        state.getResponse().setBody(bodyBlock);
     }catch (SOAPException e) {
         state.getResponse().setFailure(true);
      }
   }

    //This method must invoke the correct method on the servant based on the opcode.
   //based on opcode comming with request state it will find the methods of our class
      protected void processingHook(StreamingHandlerState state) throws Exception {
         switch (state.getRequest().getOperationCode()) {
            case getBookPrice_OPCODE:
               invoke_getBookPrice(state);
               break;
            default:
               throw new SOAPProtocolViolationException("soap.operation.unrecognized");
         }
      }
```

WSDL File



```
<?xml version="1.0" encoding="UTF-8" ?>
<definitions xmlns="http://schemas.xmlsoap.org/wsdl/"
             xmlns:tns="http://bookinfoservice.org/wsdl"
             xmlns:xsd="http://www.w3.org/2001/XMLSchema"
             xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
             name="BookInfoService"
             targetNamespace="http://bookinfoservice.org/wsdl">

  <message name="BookInfoService_getBookPrice">
    <part name="String_1" type="xsd:string" />
  </message>

  <message name="BookInfoService_getBookPriceResponse">
    <part name="result" type="xsd:float" />
  </message>

  <portType name="BookInfoService">

  <operation name="getBookPrice" parameterOrder="String_1">
     <input message="tns:BookInfoService_getBookPrice" />
     <output message="tns:BookInfoService_getBookPriceResponse" />
  </operation>

  </portType>
 - <binding name="BookInfoServiceBinding" type="tns:BookInfoService">
 - <service name="BookInfoService">
</definitions>
```

➔Whenever consumer send the request (XML) to provider over the network, then Servlet Container will receive the request and go to the web.xml file then find the URL Pattern.

➔After getting the URL pattern container will find the corresponding Servlet class then container will get JAXRPCServlet and it is vendor specific class.

➔Then container execute the JAXRPCServlet but he can't able to know what is there inside body part of the RequestMessage because actual data is wrapped inside SOAP XML and he don't know how to process SOAP XML.

➔Then container quickly go to the jaxrpc-ri-runtime.xml and searches which endpoint having same URL as servlet.

➔After getting the URL Pattern it identify the TIE Class and And handover the request to the TIE

➔ Tie class has the logic for processing the SOAP XML, and extracts the actual XML from the SOAP Body and then passes it to request SOAP serializers/De-Serializers to convert into Java Object. Then passes the Object data as input to the Service method. Once the Service method finishes processing, it returns Java Object data as response to the Tie. Now Tie converts this back into XML and then into SOAP

XML and places it into HTTP Response Message and sends back it to the Consumer.

➔Tie will first checks the OPCode by calling the processingHook() method
➔ If our class has multiple methods then by using OPCode Tie can distinguish the actual method.
➔After getting the OPCode(Operation Code) it will distinguish the actual method name and calling the invoke_getBookPrice() by passing the StremingHandlerState (Request) as input.

➔After that it will get the RequestStruct object by Type casting from StremingHandler State and it calls getTarget() to get implementation class.

➔After that container will get the parameter value from RequestStruct object by calling getter method. And then pass it inside the original method.

➔After getting the return value it will create ResponseStruct Object and call the setter on that object by passing the return value.

➔After that it binds that ResponseStruct object to SoapSerializer and send back to the consumer.

WSDL

➔WSDL stands for Web Service Description Language.
➔It is the documentation that describes the information about the provider to the outer world.
➔WSDL acts as an contract between consumer and provider.

➔WSDL describes the information about the provider in an interoperable manner.
➔The provider is an interoperable application and the data/information also interoperable so the corresponding provider must follow WSDL .
➔WSDL is an special type of XML and every xml has prolog and each xml must have one root element, for WSDL the root element is **<definitions>**
inside definitions there are 5 elements and including <definitions> 6 elements.
these are:

1.<types>
2.<message>
3.<PortType>
4.<binding>
5.<service>

One element is interrelated with each other.

Write an SEI Interface:
ex:
considering the SEI Interface write the WSDL document:

```
package com.insurance.plans;
import java.rmi.Remote;
import java.rmi.RemoteException;
public interface LicIndia extends Remote{
	Policy enroll(Member member,Plan plan)throws RemoteException;
}
```

```
public class Policy implements Serializable{
	private int policyNo;
	private int memberId;
	private String membername;
	private int planId;
	private String planName;
```

```
public class Member implements Serializable{
	private int uniqueId;
	private String name;
	private String gender;
	private String address;
```

```
public class Plan implements Serializable{
	private int planId;
	private String planName;
	private String duration;
```

<definitions name="">	**This is the root element of the WSDL document.**
<types>	**➔Types section defines the input/ output types of a Web Service method.** **➔If your web service method takes any Objects as input or returns any Object as output, those relevant complex type representations are declared under**

	<types> section of the WSDL document. ➔Either the complex types are declared directly or a Schema document is imported in the <types> section.

definations

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="InsurancePolicy"
             targetNamespace="http://licindia.com/service"
             xmlns:xsd="http://www.w3.org/2001/XMLSchema"
             xmlns:lis="http://licindia.com/service"
             xmlns:lit="http://licindia.com/type"
             xmlns="http://schemas.xmlsoap.org/wsdl/"
             xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/">
```

types

```
<types>
    <schema targetNamespace="http://licindia.com/type"
                xmlns:lis="http://licindia.com/type"
                xmlns="http://www.w3.org/2001/XMLSchema"
                xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
                xmlns:soap11-enc="http://schemas.xmlsoap.org/soap/encoding/">
    <import namespace="http://schemas.xmlsoap.org/soap/encoding/"/>
        <complexType name="member">
            <sequence>
                <element name="uniqueId" type="int"/>
                <element name="name" type="string"/>
                <element name="gender" type="string"/>
                <element name="address" type="string"/>
            </sequence>
        </complexType>
        <complexType name="plan">
            <sequence>
                <element name="planId" type="int"/>
                <element name="planName" type="string"/>
                <element name="duration" type="string"/>
            </sequence>
        </complexType>
        <complexType name="policy">
            <sequence>
                <element name="policyNo" type="int"/>
                <element name="memberId" type="int"/>
                <element name="membername" type="string"/>
                <element name="planId" type="int"/>
                <element name="planName" type="string"/>
            </sequence>
        </complexType>
    </schema>
</types>
```

Note: - In the above types section, we have an inline XSD document. And all the complexTypes are declared under target namespace.

Input Message

```
<message name="LicIndia_enroll">
      <part name="member" type="lit:member"/>
      <part name="plan" type="lit:plan"/>
</message>
```

OutPut Message

```
<message name="LicIndia_enrollResponse">
      <part name="result" type="lit:Policy"/>
</message>
```

Note: - In the above message "lit:" represents the prefix namespace under which the complex types are declared under.

<portType>	➔<portType> declares the operations of the Service. PortType exactly represents the SEI interface. ➔Generally an SEI interface doesn't provider any Implementation, rather it providers information like what are the methods and their parameters and return types. ➔Similarly portType describes about the Service operations and their input/output types. It doesn't provide transport specific information about the service

portType

```
<portType name="LicIndia">
      <operation name="enroll" parameterOrder="member plan">
        <input message="lis:LicIndia_enroll"/>
        <output message="lis:LicIndia_enrollResponse"/>
      </operation>
</portType>
```

<types>:

➔Consumer has to send required information to the provider (member data and plan data) then only the provider can process the data and fulfill the requirement.
➔The consumer sends the xml format of data as input to provider not in the object format as input.

➔Which data does the provider need as input and what output the provider is going to produce is documented in WSDL.

➔Types section defines the input/output types of a web service method.

➔A complexType represent the structure of class in Java.

Inside <types> we must be representing the parameters and the return type inside the type element.

Q.why complexType?

➔Inside complexType contain data are represent in an order and for store the data of xml we need complexType.
➔Consumers send the data to provider that information are written in WSDL. so we write complexType inside <type> element.

➔Total number of complexType depends on the number of parameters and the return/ response.

➔Always inside <types> element <xs:schema> must be there and with the schema "targetNamespace" must be there for avoid the naming conversion.

<message>

➔It represent the input and output of a web services method. the type element does not represent which are use for request and which are use for response.
➔<message> represent's the total input we are going to set and total output we are going to set.
➔All the input data are wrapped into a single message.
➔<message> contains <part> inside it which represent the parameters presents of the method and return type also.
➔Always number of input <part> depends on number of parameter in the method.
➔But in case of output <part> it always one for each method. and its name always represent as follows:

```
        <part name="result"  type="xs:string"/>
```

<portType>

➔<PortType> of my wsdl document is represent the SEI Interface of my webservices.

➔In SEI Interface we have methods and the interface name. Similarly in <portType> also a name attribute,by looking at the <portType> name we know the name of my SEI Interface name.
<portType name=""BookTicket">-- BookTicket =SEI Interface name

➔Every interface contain a method and each method have a name similarly inside <portType> a tag called <operation> which represent the details of method and its attribute is name which represent the method name.
<operation name="getTicket"   ---- getTicket =method name

➔Every method will take some parameters as an input and have some return type as output. Similarly every operation has some input and output inside it.

➔Input and output data re represent by <message>

➔All the WSDL has <message> there may be a chance of having same <message> name ,so maintain naming **convention for this we have to give typeNamespace.**
<input message=" ir:BookTicket_getTicket "/>
<output message="ir: BookTicket_getTicketResponse "/>
We have to write the targetNameSpace in the <definations> level.

WSDL is of two types

1.Abstract wsdl
2.Concrete wsdl

➔Abstract wsdl:The wsdl which contain the type, message, portType section inside it this wsdl called as Abstarct wsdl.

➔These three session generally gives information about SEI interface and method and the return type and parameter of this method and which data are for input and which data for output.

➔These three sections are not going to tell how to access the information from provider.These information gives the skeleton representation of the provider so it is called as Abstarct wsdl.

➔Concrete wsdl: The wsdl which contain all the session along with these three are called as Concrete wsdl.

<binding>	➔As <PortType> is an abstract type, it doesn't talk about transport related Aspects of the Service. ➔Binding provides implementation of the portType where in it describes **how is the data being transported between Consumer and Provider**. ➔Which protocol is being used for transmission and what is the binding protocol. ➔Along with this it describes how is the messages are exchanged between consumer and provider and in which format as well.

	➔In the binding while definition the operations we will specify soap:action for each operation. soapAction is used for resolving an input request to a method on the provider.

➔SoapAction: - While consumer sending the request to the provider, he will place soapAction (of the method we want to invoke from WSDL) in the HTTP Request header. Once the Provider has received it, he will identify the respective method to invoke based on the soapAction value in the request header.
➔It gives the information about how to access information or which transmission protocol to communicate with provider. And also tell about which message exchanging format we have to follow.

binding

```
<binding name="LicIndiaBinding" type="lis:LicIndia">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="rpc"/>
    <operation name="enroll">
        <soap:operation soapAction=""/>
        <input>
            <soap:body use="encoded"
                            encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
                            namespace="http://licindia.com/service"/>
        </input>
        <output>
            <soap:body use="encoded"
                            encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
                            namespace="http://licindia.com/service"/>
        </output>
    </operation>
</binding>
```

<binding>:

Binding talks about "how is the data being transported between consumer and provider" using PortType session.
generally binding provides 3 things
1.Transport protocol(which protocol the provider recommended to use)
2.Binding protocol
3.Message exchanging format
for transmission of (communicate) provider generally we use http protocol.<binding> have to mention that in side it.
Not only http transmission protocol we use to send thee data we udse **soap xml** on top of it . Inside soapxml we have to write the data (actual data). We have to mention in <binding> which binding protocol we use.
Binding not only talks about which protocol and which binding protocol we have to use .It talks about which message exchanging format we have to follow either rpc-encoded,rpc-literal or document-literal.
All the methods are not required the data in same format ,it differs method to method so for every operation we need different message exchanging format.
Always the binding is reffer to PortType becouse PortType says about the data which consumer send to providerand the response the provider gives back.But the binding

required these data from PortType and it talks about how the data is going to transform to provider.

<service>	➔Service acts as a Factory for create the <port> objects. Port attaches an endpoint to the Binding defined above. ➔<service> element wraps several ports under it and is responsible for creating the port objects at the consumer side.

Service

```
<service name="InsurancePolicy">
      <port name="LicIndiaPort" binding="lis:LicIndiaBinding">
        <soap:address location="REPLACE_WITH_ACTUAL_URL"/>
      </port>
</service>
```

Contract First Approach

➔This is no way different from contract last approach, all the request processing flow and other concepts (serializers/de-serializers) will still applicable even you develop the service using contract first approach. Only difference would be the way you are developing the service.

➔In web services, the contract between provider and consumer is WSDL, as it is contract first approach, the development will starts from WSDL rather than from Java. Now the developer has to write the WSDL document from which he generates the necessary java classes to expose it as service.

➔By looking at the WSDL – PortType we can create the SEI interface and based on the operation input and output types we can create method with parameters and return types. We need serializers and de-serializers to map incoming XML data to method parameters and return value of the method to XML back.

➔Working with Contract First Approach

WSDL Document

This is the root element of the WSDL document.

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions  xmlns:xs="http://www.w3.org/2001/XMLSchema"
                   xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
                   xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
                   name="trainprovider"
                   targetNamespace="http://www.irctc.co.in/rail/services"
                   xmlns:irs="http://www.irctc.co.in/rail/services"
                   xmlns:irt="http://www.irctc.co.in/rail/types">
```

➔ **If your web service method takes any Objects as input or returns any Object as output, those relevant complex type representations are declared under <types> section of the WSDL document.**

```
<wsdl:types>
        <xs:schema targetNamespace="http://www.irctc.co.in/rail/types">
          <xs:complexType name="journey-info">
            <xs:sequence>
              <xs:element name="source" type="xs:string"/>
              <xs:element name="destination" type="xs:string"/>
              <xs:element name="journer-date" type="xs:date"/>
            </xs:sequence>
          </xs:complexType>
          <xs:complexType name="passenger-info">
            <xs:sequence>
              <xs:element name="unique-id" type="xs:string"/>
              <xs:element name="name" type="xs:string"/>
              <xs:element name="age" type="xs:int"/>
              <xs:element name="gender" type="xs:string"/>
            </xs:sequence>
          </xs:complexType>
           <xs:complexType name="ticket">
             <xs:sequence>
               <xs:element name="pnr-no" type="xs:string"/>
               <xs:element name="train-no"  type="xs:string"/>
               <xs:element  name="seat" type="xs:int"/>
               <xs:element name="coach" type="xs:string"/>
             </xs:sequence>
           </xs:complexType>
        </xs:schema>
 </wsdl:types>
```

➔**It represents the input and output of a web services method. The type element does not represent which is use for request and which is use for response.**

➔**<message> contains <part> inside it which represent the parameters presents of the method and return type also.(message talks about Input Method and output Method parameters and return types)**

```
<wsdl:message name="TrainProvider_bookTicket_Request">
 <wsdl:part name="journey-info"  type="irt:journey-info"/>
 <wsdl:part name="passenger-info"  type="irt:passenger-info"/>
</wsdl:message>

<wsdl:message name="TrainProvider_bookTicket_RequestResponse">
 <wsdl:part name="result" type="irt:ticket"/>
</wsdl:message>
```

➔Every method will take some parameters as an input and have some return type as output. Similarly every operation has some input and output inside it.(What to access)

```
<wsdl:portType name="TrainProvider">
 <wsdl:operation name="bookTicket">
  <wsdl:input message="irs:TrainProvider_bookTicket_Request"/>
  <wsdl:output message="irs:TrainProvider_bookTicket_RequestResponse"/>
 </wsdl:operation>
</wsdl:portType>
```

➔Binding talks about "how is the data being transported between consumer and provider" using portType session.(Binding talks about how to access)

```
<wsdl:binding name="trainproviderSOAPBinding" type="irs:TrainProvider">
 <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
 <wsdl:operation name="bookTicket">
  <soap:operation soapAction="http://www.irctc.co.in/rail/services/bookTicket"/>
  <wsdl:input>
   <soap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
              namespace="http://www.irctc.co.in/rail/services" use="encoded"/>
  </wsdl:input>
  <wsdl:output>
   <soap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
              namespace="http://www.irctc.co.in/rail/services" use="encoded"/>
  </wsdl:output>
 </wsdl:operation>
</wsdl:binding>
```

➔Service acts as a Factory for create the <port> objects. Port attaches an endpoint to the Binding. (**Port talks about where to access)**

```
<wsdl:service name="trainprovider">
  <wsdl:port binding="irs:trainproviderSOAPBinding" name="trainproviderSOAPPort">
    <soap:address location="http://www.example.org/"/>
  </wsdl:port>
</wsdl:service>

</wsdl:definitions>
```

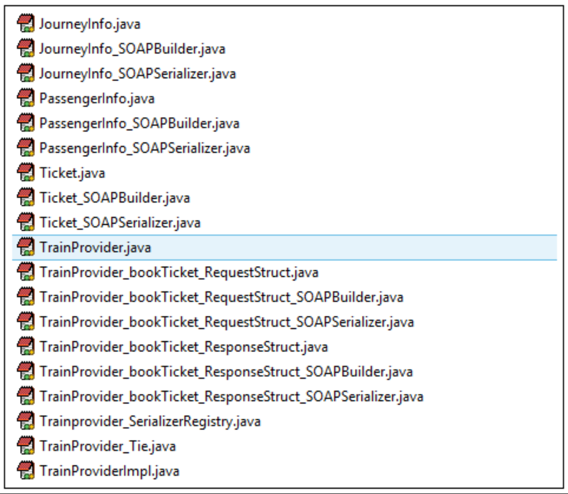
➔Instead of developer writing the above classes, we can run the wscompile tool which is shipped as part of JWSDP by giving input as WSDL. Now the wscompile will loads the WSDL document and reads various elements of WSDL and generates the required classes to build the Web service. Following are the classes generated when we run wscompile.

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration
  xmlns="http://java.sun.com/xml/ns/jax-rpc/ri/config">
  <wsdl
    location="./WebContent/WEB-INF/trainprovider.wsdl"
    packageName="com.tp.service">
  </wsdl>
</configuration>
```

How to run the wscompile tool using command?

Set the path (OR) set the environment variable

C:\>set path=%path%;C:\Sun\jwsdp-2.0\jaxrpc\bin
Go to Project Location
C:\>cd workspace\TrainProvider
Run wscompile with switches
C:\workspace\TrainProvider>**wscompile -d src -gen:server -keep -verbose -model bsmodel-rpc-enc.xml.gz WebContent\WEB-INF\config.xml**



```
JourneyInfo.java
JourneyInfo_SOAPBuilder.java
JourneyInfo_SOAPSerializer.java
PassengerInfo.java
PassengerInfo_SOAPBuilder.java
PassengerInfo_SOAPSerializer.java
Ticket.java
Ticket_SOAPBuilder.java
Ticket_SOAPSerializer.java
TrainProvider.java
TrainProvider_bookTicket_RequestStruct.java
TrainProvider_bookTicket_RequestStruct_SOAPBuilder.java
TrainProvider_bookTicket_RequestStruct_SOAPSerializer.java
TrainProvider_bookTicket_ResponseStruct.java
TrainProvider_bookTicket_ResponseStruct_SOAPBuilder.java
TrainProvider_bookTicket_ResponseStruct_SOAPSerializer.java
Trainprovider_SerializerRegistry.java
TrainProvider_Tie.java
TrainProviderImpl.java
```

```java
package com.tp.service;

public interface TrainProvider extends java.rmi.Remote {
   public Ticket bookTicket(JourneyInfo journeyInfo, PassengerInfo passengerInfo) throws RemoteException;
}
```

SEI Interface	Will be generated based on portType in WSDL
Input/Output classes	➔You SEI Interface methods will take parameters and by looking at the PortType operations we will know what are input/output elements. ➔Based on this we can find their relevant XSD types and can generate classes representing input/output parameters of the SEI Methods.
Request/Response Struct, Serializers and Deserializers	These classes are required to convert the input XML to Object and Object back to XML

➔As all the classes are available to build the Service, we need to write now the Implementation class implementing the SEI interface. Then we need to write the jaxrpc-ri.xml, which attaches url to the endpoint.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<webServices
   xmlns="http://java.sun.com/xml/ns/jax-rpc/ri/dd"
   version="1.0"
   targetNamespaceBase="http://www.irctc.co.in/rail/services"
   typeNamespaceBase="http://www.irctc.co.in/rail/types"
   urlPatternBase="/tp">
   <endpoint
      name="TrainProvider"
      displayName="TrainProvider  Service"
      description="Railway service"
      wsdl="/WEB-INF/trainprovider.wsdl"
      interface="com.tp.service.TrainProvider"
      implementation="com.tp.service.TrainProviderImpl"
       model="/WEB-INF/tpmodel-rpc-enc.xml.gz"/>
   <endpointMapping
      endpointName="TrainProvider"
      urlPattern="/tp"/>
</webServices>
```

How to run the wsdeploy tool using command?

➔Put the WSDL File and Model File inside WEB-INF along with jaxrpc-ri.xml.
➔Set the path (OR) set the environment variable
C:\>set path=%path%;C:\Sun\jwsdp-2.0\jaxrpc\bin
➔Create a directory under C:\jax-rpc-wars
➔Export your Eclipse Project as WAR file and place it in side
C:\jax-rpc-wars\TrainProvider.war

```
C:\jax-rpc-wars>wscompile –o target.war  TrainProvider.war
```

➔The above command reads the contents of TrainProvider.war and identifies the web.xml and jaxrpc-ri.xml.
➔It reads the contents of web.xml and updates it with some more additional configuration.
➔Along with this it uses the jaxrpcri. xml as input file and generates web service deployment descriptor jaxrpc-ri-runtime. xml which contains the configuration related to service we are exposing.
➔These two files will be generated and placed inside the target.war. Now we can either directly deploy the target.war on the Application Server.
➔ Or extract the target.war and copy web.xml and jaxrpc-ri-runtime.xml and can paste our eclipse project inside WEB-INF DIRECTORY and deploy from eclipse as well.

Jaxrpc-ri-runtime.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<endpoints xmlns='http://java.sun.com/xml/ns/jax-rpc/ri/runtime' version='1.0'>
  <endpoint
    name='TrainProvider'
    interface='com.tp.service.TrainProvider'
    implementation='com.tp.service.TrainProviderImpl'
    tie='com.tp.service.TrainProvider_Tie'
    model='/WEB-INF/tpmodel-rpc-enc.xml.gz'
    wsdl='/WEB-INF/trainprovider.wsdl'
    service='{http://www.irctc.co.in/rail/services}trainprovider'
    port='{http://www.irctc.co.in/rail/services}trainproviderSOAPPort'
    urlpattern='/tp'/>
</endpoints>
```

web.xml

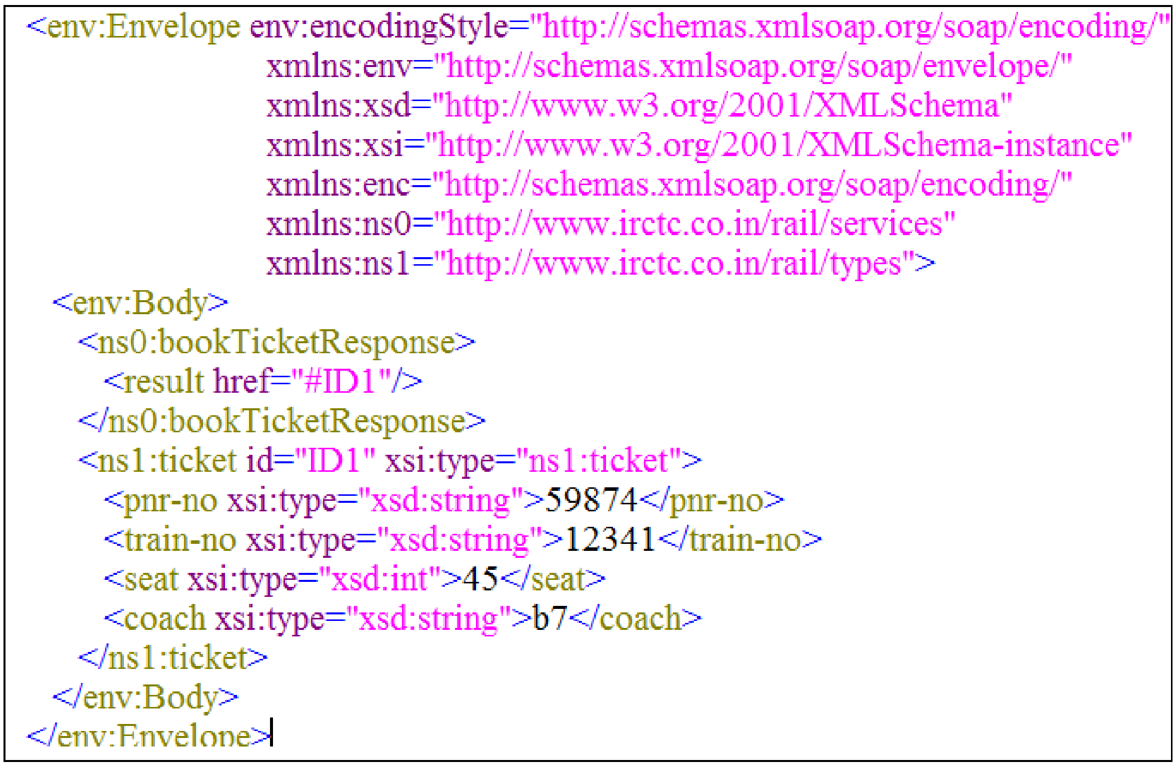
```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://java.sun.com/xml/ns/javaee"
              xmlns:web="http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
              xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
              id="WebApp_ID" version="2.5"
              xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd">
  <display-name>TrainProviderWeb</display-name>
  <listener>
<listener-class>com.sun.xml.rpc.server.http.JAXRPCContextListener</listener-class>
</listener>
<servlet>
<servlet-name>TrainProvider</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>TrainProvider</servlet-name>
<url-pattern>/tp</url-pattern>
</servlet-mapping>
<welcome-file-list>
   <welcome-file>tp</welcome-file>
  </welcome-file-list>
</web-app>
```

➔To check the request and response we have to use an application **SOAP-UI**
➔**Open SOAP-UI and create a Project**
➔ **And give the WSDL file Link to the SOAP-UI**

Request From SOAP

```
<soapenv:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
                  xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
                  xmlns:ser="http://www.irctc.co.in/rail/services">
  <soapenv:Header/>

  <soapenv:Body>
    <ser:bookTicket soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
      <journey-info xsi:type="typ:journey-info" xmlns:typ="http://www.irctc.co.in/rail/types">
        <source xsi:type="xsd:string">Hyderabad</source>
        <destination xsi:type="xsd:string">Bangalore</destination>
        <journer-date xsi:type="xsd:date">2017-02-22</journer-date>
      </journey-info>
      <passenger-info xsi:type="typ:passenger-info" xmlns:typ="http://www.irctc.co.in/rail/types">
        <unique-id xsi:type="xsd:string">I65523</unique-id>
        <name xsi:type="xsd:string">Dhananjaya</name>
        <age xsi:type="xsd:int">27</age>
        <gender xsi:type="xsd:string">male</gender>
      </passenger-info>
    </ser:bookTicket>

  </soapenv:Body>
</soapenv:Envelope>
```

Response from Soap UI



```xml
<env:Envelope env:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
                        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://www.irctc.co.in/rail/services"
                        xmlns:ns1="http://www.irctc.co.in/rail/types">
  <env:Body>
    <ns0:bookTicketResponse>
      <result href="#ID1"/>
    </ns0:bookTicketResponse>
    <ns1:ticket id="ID1" xsi:type="ns1:ticket">
      <pnr-no xsi:type="xsd:string">59874</pnr-no>
      <train-no xsi:type="xsd:string">12341</train-no>
      <seat xsi:type="xsd:int">45</seat>
      <coach xsi:type="xsd:string">b7</coach>
    </ns1:ticket>
  </env:Body>
</env:Envelope>
```

Q.When and where to use contract last approach?

➔There may be a chance of when one developer was assigned to a project at that time the project was under development and several releases's also completed.

➔We may develop this web application in contract first approach but we have to write the interface and implementation classes and binding class once again and deploy the code for retesting and all these process gives huge financial loss and time also so don't go for contract first approach .

➔Then it's better to go with contract last approach because the functionality and business logic and related classes are already written. Rather than going for developing the web application from the scratch level go for contract last approach.

Q.When and where you use contract first approach?

➔When we join in a project development team suppose it is start from scratch level .
➔And the phase 1 of the project was out to production team and i join with the team in phase 2.
➔In phase 2 some new module added and there was already the business requirement document is there so I get a chance to write the interface for the particular module and write the implementation classes and all these process, so I got a chance of working in contract last approaches.
➔Which web application is develop from scratch level then we use contract first approach.

Q.What is webservice business requirement document?

➔Generally for the development of an web service which business logic we have to follow/apply and what are the interfaces and what are the implementation classes are required are finalized/described greater level by the analysts and provide in a document manner called as webservice requirement document.
➔We know what we have developed with the help of webservice requirement document. all the standard convention and the name of the port name, operation and the element name with the uppercase and lowercase also written in business requirement document.

Q.Where to use opcode?

➔If there are two methods are there in our class.
➔Then how JAX-RPC SI implementation distinguish them so that OP code comes into picture.
➔OP code is not a standard specific, it differ vendor to vendor, it specify to only jax-rpc si implementation, so JAX-RPC SI implementation internally used OPcode to distinguish between multiple methods , The other vender's implementation class are not use the OPcode .

Q.What is SOAPAction?

SOAP Action is a standard specification to represent the index of methods. It supports for all the vender specific implementation .
➔Due to the misunderstandings of OP code SOAP Action come into the picture to overcome from this problem.

Q.How the request method call resolved?

➔In case of contract last approach we generate the WSDL at last.
➔So we don't specify the SOAP Action which is the first method and which is second method.
➔So jax-rpc SI implementation support two ways of request method call (SOAP Action and OP code).
➔In contract last approach by default OPcode will generated by the jax-rpc SI implementation.
➔WE can also modify the OP code to SOAP Action manually in contract last approach after generating the WSDL.

Q.Why we place WSDL under WEB-INF?

➔We are placing the WSDL Document inside WEB-INF because of following reasons.
➔Because our class should not access directly by class name because if in future class name will change then consumer will modify his code multiple times.
➔So place the WSDL inside WEB-INF and provide a URL to access, So that if provider class name will change then no consumer needs not to modify the code.

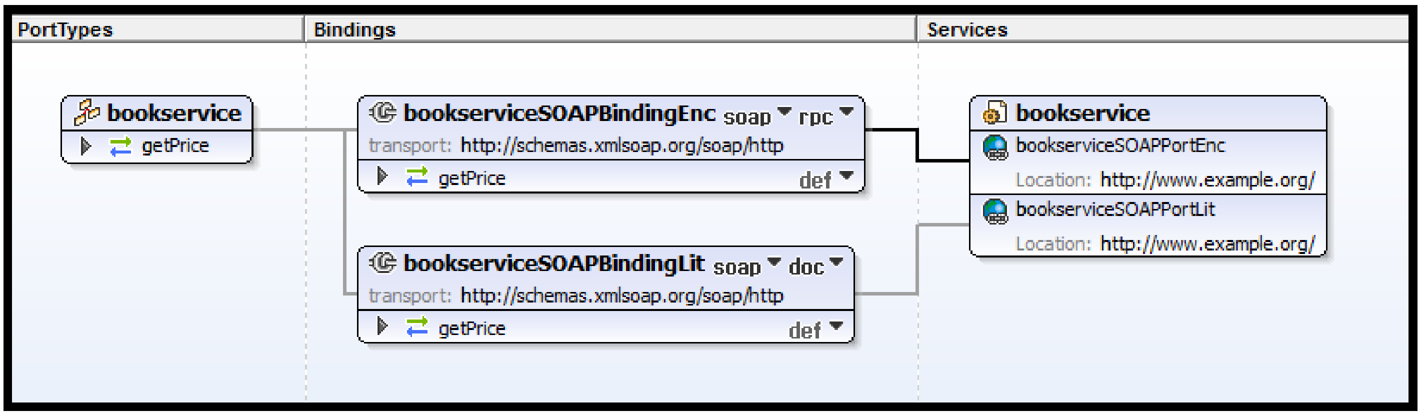
Permutations and combinations you have to try

➔Change the parameter name of SEI interface and check whether it is working or not.
➔Change the logic inside implementation method of SEI Interface and check whether it is working or not.
➔Change the data type of SEI Interface methods and check whether it is working or not.
➔Change the return type of SEI interface methods and check whether it is working or not.

➔Try with two overloaded methods and check whether it is working or not.
➔Don't create the Model file and check whether it is working or not.
➔Change the SEI interface implementation class name and check whether it is working or not.
➔Try with two Message Exchanging Format with two different WSDL.
➔ Try with two Message Exchanging Format with one WSDL.

Working with Two Message Exchanging Format

```
package com.bs.services;

public interface Bookservice_PortType extends java.rmi.Remote {
    public float getPrice(java.lang.String isbn) throws
        java.rmi.RemoteException;
}
```


PortTypes
Bindings
Services
bookservice
getPrice
bookserviceSOAPBindingEnc soap rpc
transport: http://schemas.xmlsoap.org/soap/http
getPrice def
bookserviceSOAPBindingLit soap doc
transport: http://schemas.xmlsoap.org/soap/http
getPrice def
bookservice
bookserviceSOAPPortEnc
Location: http://www.example.org/
bookserviceSOAPPortLit
Location: http://www.example.org/

➔First you have to write two binding with two different different Message Exchanging Format in a single WSDL.
➔Then you have to write two types of port as per binding.

```
<wsdl:binding name="bookserviceSOAPBindingEnc" type="ass:bookservice">
  <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
  <wsdl:operation name="getPrice">
    <soap:operation soapAction="http://www.amazon.co.in/sales/services/getPrice"/>
    <wsdl:input>
      <soap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
                 namespace="http://www.amazon.co.in/sales/services"
                 use="encoded"/>
    </wsdl:input>
    <wsdl:output>
      <soap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
                 namespace="http://www.amazon.co.in/sales/services"
                 use="encoded"/>
    </wsdl:output>
  </wsdl:operation>
</wsdl:binding>
```

```
<wsdl:binding name="bookserviceSOAPBindingLit" type="ass:bookservice">
  <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
  <wsdl:operation name="getPrice">
    <soap:operation soapAction="http://www.amazon.co.in/sales/services/getPrice"/>
    <wsdl:input>
      <soap:body namespace="http://www.amazon.co.in/sales/services"
                 use="literal"/>
    </wsdl:input>
    <wsdl:output>
      <soap:body namespace="http://www.amazon.co.in/sales/services"
                 use="literal"/>
    </wsdl:output>
  </wsdl:operation>
</wsdl:binding>
```

```
<wsdl:service name="bookservice">

 <wsdl:port binding="ass:bookserviceSOAPBindingEnc" name="bookserviceSOAPPortEnc">
   <soap:address location="http://www.example.org/"/>
 </wsdl:port>

 <wsdl:port binding="ass:bookserviceSOAPBindingLit" name="bookserviceSOAPPortLit">
   <soap:address location="http://www.example.org/"/>
 </wsdl:port>

</wsdl:service>
```

➔After that run wscompile tools 2 times whenever you run the wscompile tools to generate the binding class for rpcencoded you have to comment the rpcliteral port in side service section.

```
C:\workspace\BookService>wscompile –d src –gen:server –keep –verbose –model bsmodel-rpc-enc.xml.gz   WebContent\WEB-INF\config.xml
```

➔After that whenever you run the wscompile tools to generate the binding class for rpcliteral you have to comment the rpcencoded port in side service section.

```
C:\workspace\BookService>wscompile –d src –gen:server –keep –verbose -f:rpcliteral –model bsmodel-rpc-lit.xml.gz   WebContent\WEB-INF\config.xml
```

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration
  xmlns="http://java.sun.com/xml/ns/jax-rpc/ri/config">
  <wsdl
    location="WebContent/WEB-INF/bookservice.wsdl"
    packageName="com.bs.servicesEnc">
  </wsdl>
</configuration>
```

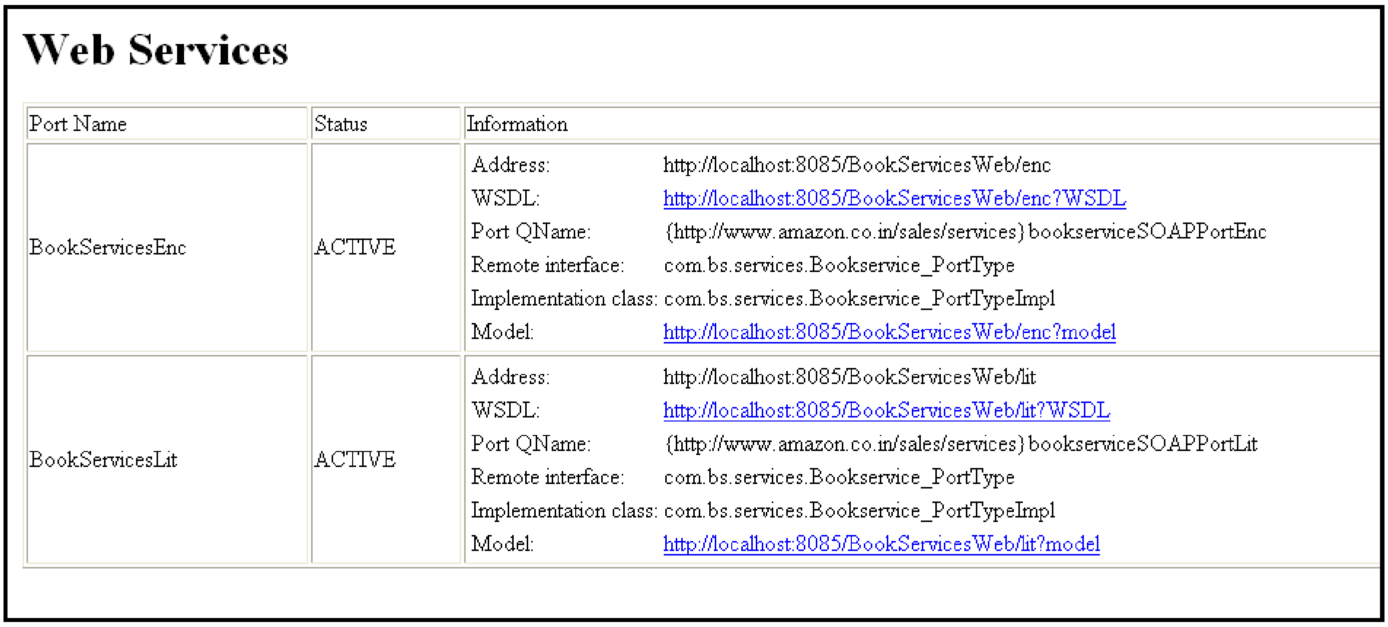
```
<?xml version="1.0" encoding="UTF-8"?>
<configuration
  xmlns="http://java.sun.com/xml/ns/jax-rpc/ri/config">
  <wsdl
    location="WebContent/WEB-INF/bookservice.wsdl"
    packageName="com.bs.servicesLit">
  </wsdl>
</configuration>
```

➔Then you have to run wsdeploy tools two times because JAXRPC-RI.XML allows at a time one endpoint to generate JAXRPC-RI-RUNTIME.XML . Then you have to append both the endpoint in one JAXRPC-RI-RUNTIME.XML.

```
<?xml version="1.0" encoding="UTF-8"?>
<endpoints xmlns='http://java.sun.com/xml/ns/jax-rpc/ri/runtime' version='1.0'>
 <endpoint
   name='BookServicesEnc'
   interface='com.bs.services.Bookservice_PortType'
   implementation='com.bs.services.Bookservice_PortTypeImpl'
   tie='com.bs.services.Bookservice_PortType_Enc_Tie'
   model='/WEB-INF/bsmodel-rpc-enc.xml.gz'
   wsdl='/WEB-INF/bookservice.wsdl'
   service='{http://www.amazon.co.in/sales/services}bookservice'
   port='{http://www.amazon.co.in/sales/services}bookserviceSOAPPortEnc'
   urlpattern='/enc'/>

   <endpoint name="BookServicesLit"
               interface="com.bs.services.Bookservice_PortType"
               implementation="com.bs.services.Bookservice_PortTypeImpl"
               tie="com.bs.services.Bookservice_PortType_Lit_Tie"
               model="/WEB-INF/bsmodel-rpc-lit.xml.gz"
               wsdl="/WEB-INF/bookservice.wsdl"
               service="{http://www.amazon.co.in/sales/services}bookservice"
               port="{http://www.amazon.co.in/sales/services}bookserviceSOAPPortLit"
               urlpattern="/lit" />

</endpoints>
```

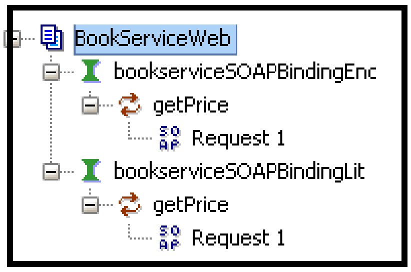
```xml
<servlet>
  <servlet-name>BookServicesEnc</servlet-name>
  <servlet-class>com.sun.xml.rpc.server.http.JAXRPCServlet</servlet-class>
  <load-on-startup>1</load-on-startup>
</servlet>
<servlet-mapping>
  <servlet-name>BookServicesEnc</servlet-name>
  <url-pattern>/enc</url-pattern>
</servlet-mapping>
```

```xml
<servlet>
  <servlet-name>BookServicesLit</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>BookServicesLit</servlet-name>
  <url-pattern>/lit</url-pattern>
</servlet-mapping>
```



Web Services

Port Name	Status	Information
BookServicesEnc	ACTIVE	Address: http://localhost:8085/BookServicesWeb/enc WSDL: http://localhost:8085/BookServicesWeb/enc?WSDL Port QName: {http://www.amazon.co.in/sales/services}bookserviceSOAPPortEnc Remote interface: com.bs.services.Bookservice_PortType Implementation class: com.bs.services.Bookservice_PortTypeImpl Model: http://localhost:8085/BookServicesWeb/enc?model
BookServicesLit	ACTIVE	Address: http://localhost:8085/BookServicesWeb/lit WSDL: http://localhost:8085/BookServicesWeb/lit?WSDL Port QName: {http://www.amazon.co.in/sales/services}bookserviceSOAPPortLit Remote interface: com.bs.services.Bookservice_PortType Implementation class: com.bs.services.Bookservice_PortTypeImpl Model: http://localhost:8085/BookServicesWeb/lit?model



```
BookServiceWeb
  bookserviceSOAPBindingEnc
    getPrice
      Request 1
  bookserviceSOAPBindingLit
    getPrice
      Request 1
```

Q.How does the provider will distribute to consumer?

➔We can't provide the WSDL to the consumer directly in a live environment.
➔There are other environments are there such as QA Environment, number of stages and Production environment.
➔Each environment has its URL, provider is going to share these URL to consumer to use the provider's environment.
➔Suppose if the consumer is in stage then the provider will give the stage URL and if the consumer is in Q/A then provider will give the URL of Q/A to access it.

Q.How does the bugs are reported within your project.

➔Provider side people are not going to raise the bug directly first they will investigation the issue, whether that issue is real or not, after investigation if they found the bug is real, then the provider side raise a bug.
➔Provider is going to give a bug number of the particular bug and send the bug number to consumer which is fist raise by the consumer.
➔Then consumer will match their bug number in the bug management tool and the bug management tool is track the bug till bug has fixed in provider side.

Q.What is the challenging task you have done in your project?

Ans-1

➔At the initial stage of my job when I am start working in web service, I have a little knowledge in web services and also don't in how to handling them, due to lacks of proper knowledge.
➔ I have going through lot of technical problem and even lot of manual problem also in my carrear.I can share one of my experience/challenge if you allow,
➔When we develop provider and after develop we distribute it to the consumer.
➔When consumer develops their application in their side, there solution is purely dependent with provider side so if they found any error then first they inform to the provider which I would not understand how the work is going on in integration system.
➔But I thought that whenever consumer send a bug report I have to reproduce that bug and resolve the problem which takes lot of time due to my improper knowledge.
➔I don't know that when a bug raise first of all we have to investigate it whether it is proper or not and the cause of the issue.
➔After 2-3 days by the help of a senior I knew that when there is a bug that issue by the consumer.
➔We encourage them to access the provider using the SOAP UI and what they send based on the response if the SOAP UI gives wrong output we told to consumer to send the screen short(we have to know the end point url what they enter is correct or not) of the inputs and outputs.
➔With that input and output we check it in our environment if the both sides output we get same by using the SOAP UI.
➔If we get same output we quickly go to the database and checks the input data are there or not, if the data are there then we can say the data which you get are present in database there is no problem in provider side.
➔This is the challenging task I have, even I can't forget those days due to that at that time all the people are pointing to me to fix the bug because the bug that has raised at consumer side is in severity-1.

Ans-2

➔First test done in provider side by using SAP UI. (By passing input and check the output) which takes lot of time for that we have to write test cases. We have to run the consumer using test cases to check my provider.
➔All the environment are connected with each other with an internet (private network) which is called **VPN (virtual private network)**.

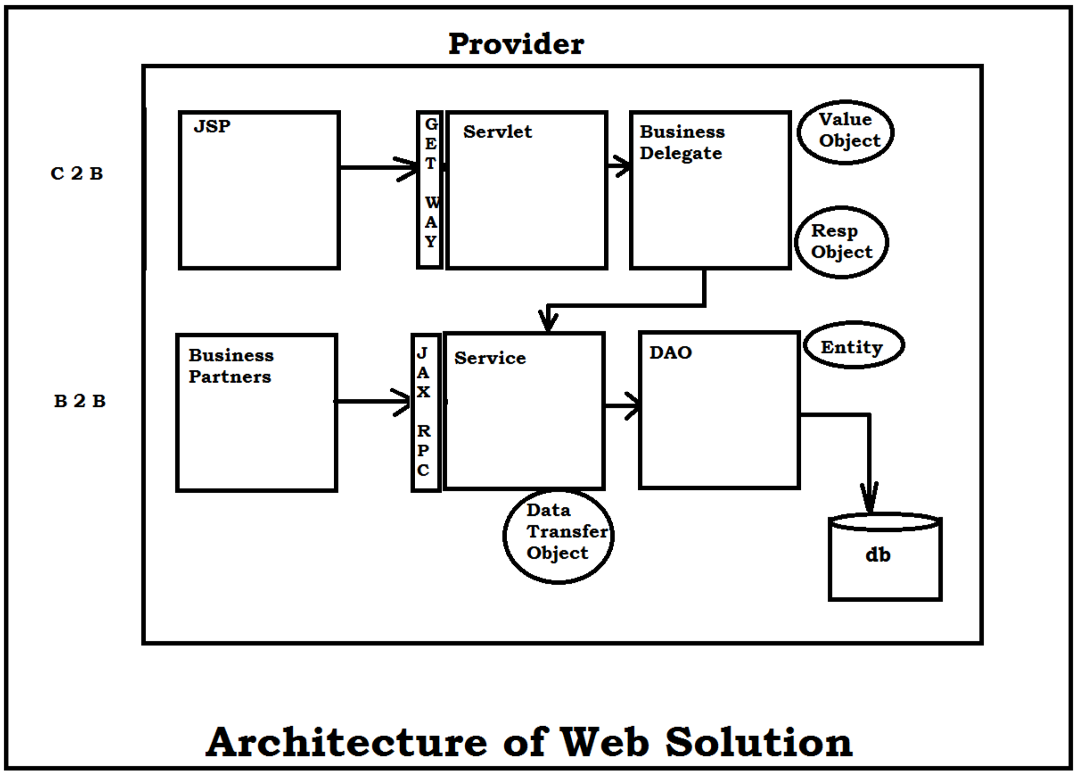
➔We are working for clients and all the systems and database and repository are with clients and connected with VPN.
➔We access the client machine by using user id and password of the VPN (**CISCO VPN**) what it assigned to us, It protect from other outer side to access the VPN.
➔ Commonly problem: Unreachable Port exception (because of improper connection to VPN) this is one problem that initially I don't know how to use VPN.
➔This not a technical task but this is the things happened with me, I can remember those days because the bug that has raised at consumer side is in severity-1 and due to lacks of knowledge it took 2-3 days which is a common problem.

Q.Did you ever develop cross technology or platform providers? In such case what is your experience?

➔If our consumer use .Net technology or any other technology in their application, it's not necessary that we have to develop the provider in the same technology.
➔We have to pass the WSDL document, if we provide WSDL our job is completed.
➔It's not our business that we have to write logic in their technology to make the consumer access from provider.
➔The only thing provider side developer can do is give some set of input and output list in terms of test cases whether it is a cross technology or not.
➔Because developer follows the terms and rules of Business requirement document which has given by business analyst.
➔After develop the provider part we have to give it to the consumer in two ways

1. WSDL
2. User guide/developer guide

➔WSDL only contains the information of provider in technical way.
➔User guide/developer guide contains all the rest information such as which method gives what output and required which kind of output, what is the messaging format and how to access the component of consumer?
➔Generally we don't give any building solution to consumer ,provider side only give the input for building the solution ,whether it is cross platform or same platform technology.



Architecture of Web Solution

Consumer

JAX-RPC API supports developing provider and consumer.
JAX-RPC supports three types of consumers:

1. Stub based consumer
2. Dynamic Proxy consumer (DP)
3. Dynamic Invocation Interface consumer(DII)

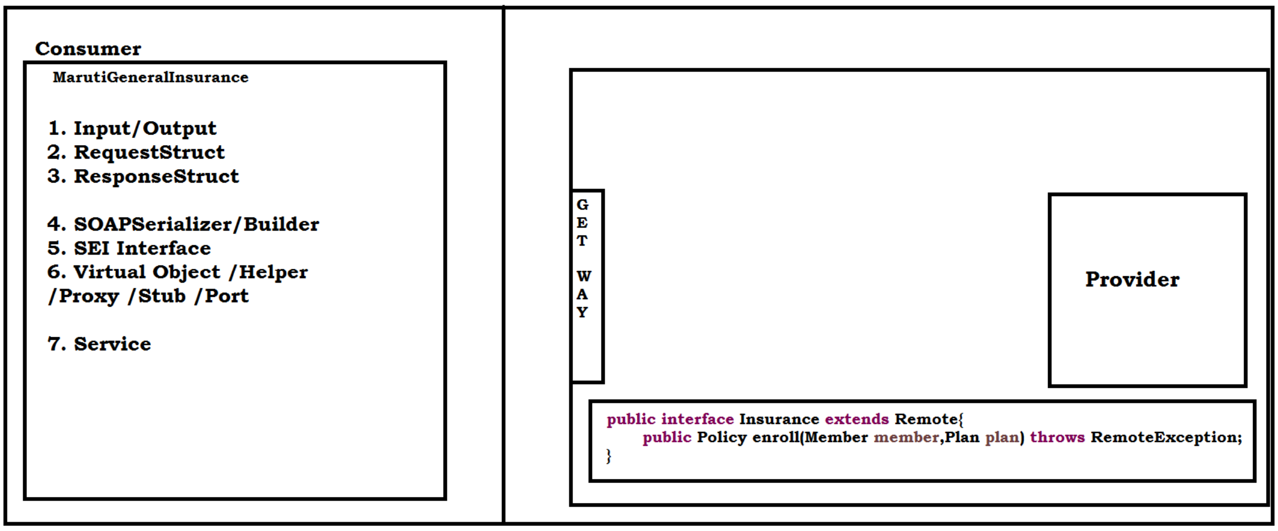
Interview Question:

1.How far you working on consumer?Can you tell me the development steps?

➔Actually the JAX-RPC API support multiple consumer out of which the most reputedly use consumer is Stub based consumer so I worked on Stub based consumer in my project.

Interviewer:"what there are multiple consumer provided by JAX-RPC, Can you just tell what the others are"

➔Yaa!!JAX-RPC is designed for fulfilling various requirement of working on consumer side. There are multiple types of consumer are addressed by JAX-RPC API,but most of us are working with Stub based consumer, we have also working on Stub based consumer .
➔There is something called DP and DII I just know about them and due to my personal interest I go through highlevely.And couple of time I experiment myself and practice on it.
Interviewer: OOo… Can you explane me what is DP and DII? Explane In and out of DP and DII.



```
public interface Insurance extends Remote{
      public Policy enroll(Member member,Plan plan) throws RemoteException;
}
```

Stub based consumer

➔We create additional classes in consumer side, which classes are called Stub classes/additional classes, with the help of these classes only we talk with the provider. That's why it is called stub based consumer.

Let's take a use case

➔We have a provider "ICICI Lombard" who provides Insurance and we have one of the consumers "Maruti General Insurance".
➔So to access our provider we need the SEI Interface of provider from which consumer know what are the input required to our provider and what our provider give to consumer as result.

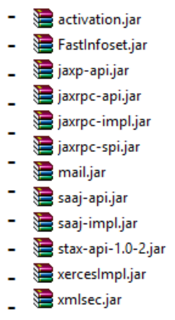
➔For this our consumer required information of ICICI Lombard but due to security issue provider does not give its class or source code or database to consumer to access.
➔All the business information are present with ICICI Lombard and provider knows which information's are required for accessing the provider.
➔Provider creates a object of the respected class and don't give the reference of the object to consumer .Who ever need the object can come and take the reference of from provider so **gate way** comes into picture.

Steps to create stub based consumer

➔First of all run the Provider Applicaton.
➔create a java project
➔create a lib folder inside your project and paste all the required jars.
➔create an etc folder and put config.xml file.
➔JAX RPC provide wscompile tool which is responsible for generating binding class.
➔wscompile tool take the config.xml as input.
➔Inside the config.xml write the location of your WSDL.
➔wscompile tool creates the helper class, which is Stub based on number of port present in WSDL.

InsuranceProviderImpl
-src
-etc
-config.xml

-lib

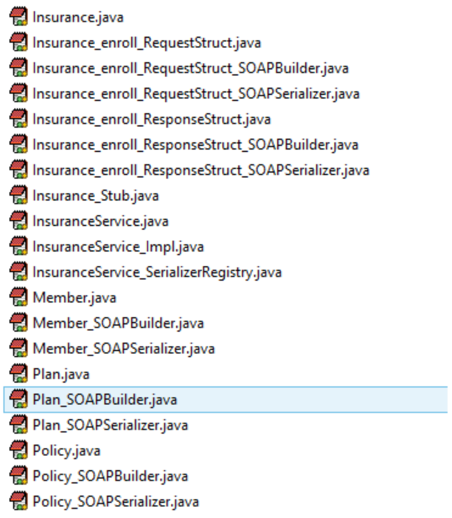


- activation.jar
- FastInfoset.jar
- jaxp-api.jar
- jaxrpc-api.jar
- jaxrpc-impl.jar
- jaxrpc-spi.jar
- mail.jar
- saaj-api.jar
- saaj-impl.jar
- stax-api-1.0-2.jar
- xercesImpl.jar
- xmlsec.jar

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration
  xmlns="http://java.sun.com/xml/ns/jax-rpc/ri/config">
  <wsdl
    location="http://localhost:8085/InsuranceProvider/enroll?WSDL"
    packageName="com.maruti.generalInc">
  </wsdl>
</configuration>
```

1.Open Command Prompt
2. Go to the Project location
c:\workspace\InsuranceProviderImpl>**wscompile -d src -gen:server -keep -verbose etc\config.xml**

com.maruti.generalInc



Insurance.java
Insurance_enroll_RequestStruct.java
Insurance_enroll_RequestStruct_SOAPBuilder.java
Insurance_enroll_RequestStruct_SOAPSerializer.java
Insurance_enroll_ResponseStruct.java
Insurance_enroll_ResponseStruct_SOAPBuilder.java
Insurance_enroll_ResponseStruct_SOAPSerializer.java
Insurance_Stub.java
InsuranceService.java
InsuranceService_Impl.java
InsuranceService_SerializerRegistry.java
Member.java
Member_SOAPBuilder.java
Member_SOAPSerializer.java
Plan.java
Plan_SOAPBuilder.java
Plan_SOAPSerializer.java
Policy.java
Policy_SOAPBuilder.java
Policy_SOAPSerializer.java

➔Create a Test class
➔To call provider specific method we need the port.
➔Service is the factory for the port, so through the help of service we can get port.
➔After getting the port store it in SEI interface and call the method on that.

```
public class Test {
   public static void main(String[] args) throws ServiceException, RemoteException {
      InsuranceService service=new InsuranceService_Impl();
      Insurance port=service.getInsuranceSOAPPort();


      Member member=new Member();
      member.setUniqueNo("Id-1001");
      member.setMemberName("Dhananjaya");
      member.setMemberAddress("Hyderabad");
      member.setGender("male");


      Plan plan=new Plan();
      plan.setPlanName("Jivan Saral");
      plan.setTenure("35");

      Policy policy=port.enroll(member, plan);

      System.out.println(policy);
   }
}
```

Internals

How to talk to the object? (not get the object)

➔Consumer not perform the business operation ,consumer only gives the input data to the provider.
➔For knowing which input provider required and which output you need .for this the consumer need <types> and <portType> of the WSDL.
➔Consumer has to send the required data and information of the port in a SOAP represent xml not in object format.
➔Which contains RequesStruct, ResponseStrut,SOAPSerializer/builder, Message exchanging format and Message exchanging Patten for communicate with provider. Apart from this consumer need the operation logic in a separate class
➔RequestStruct represent all the inputs that are going to send to provider.
➔ResponseStruct represent all the outputs provider going to give as response.
➔SOAP is used for communicating these Request and Response Struct to the provider and consumer respectively in a soap-xml format.
➔Binding classes are there in both consumer and provider side.
➔We have Request and Response Struct and SOAPSerializer/Builder but for communicating with provider we need the distributed object of provider.
➔We have to write a transport logic to communicate to provider ,in which class we write these logic called **helper class.**
➔Generally all the other binding classes are use the helper class to communicate with provider, they don't know any information about the provider and treat the helper class as provider or **Proxy** .
➔**Helper class** is communicate with the provider directly. So all the binding class are need helper class's help and treat it as an object and it is also called as **Virtual object** .It

gives the business logic to all the classes of consumer. we can also called as **port** to the helper class.

Q.What is there in side Virtual object?

➔It contains the relevant logic to communicate with the provider.
➔And provider contains the business logic and that will be distributed to multiple methods.
➔Every method present inside the SEI interface have required different information and different message exchanging format as per there requirement.
➔For every method in provider there is relevant method present in helper.
Inside our consumer there is also an SEI Interface and inside this interface helper class/proxy/virtual object is present. which is an implementation class of SEI interface.

Flow of communication inside consumer side

➔Consumer side classes are communicating to SEI interface.(not to original class of provider side)
➔By creating object of helper and assign to the SEI interface ,call the methods of SEI interface and it will call the method of proxy class.
➔Helper will talk to serializer and desirializer takes the data and convert to xml.
➔Opens a connection to provider send the data at SOAP over xml.
➔Provider send the response back to helper as SOAP Desriliazation.

Q.How to access the service of the port?

➔First deside which service you want to talk to,fro the service know the port,and from the port we know the binding and from the binding we understand transport logic and endpoint url from which we access the service.

Q.How many virtual objects will be there in consumer side?

➔It Depends on number of ports.

Dynamic Proxy (DP)

➔In this type of consumer, we don't need to generate the binding classes or stub/virtual object/proxy/port at the design time with the help of vender provided tools.
➔The Stub will get generated on runtime, that's why this type of consumer called as Dynamic proxy.

➔For communicate with provider first we have to check whether all the
- ➔Input/Output are there or not
- ➔Reuquest/Response Struct
- ➔Serializers/De-serializer/Builder
- ➔SEI interface
- ➔Service
- ➔Stub

Are there or not, without these classes we can't communicate to provider.

➔But in case of Dynamic proxy we generate the proxy/stub at runtime so proxy is not required at the consumer side.

➔All the Input and output, Request/Response Stuct, Serializer/De-serializer are depends on proxy, In Dynamic proxy the Stub will generate at runtime so all these classes are also generated at runtime.

To start a Dynamic Proxy based consumer what we need?

Ans:

➔Input classes and output classes, We get the input and output type by looking at the complex Type.
➔SEI interface: by looking at the port Type we can write the SEI interface.
➔We don’t need any tools and we don't need to generate any classes.
➔Use low-level API which is provided by JAX-RPC to build runtime proxy (Dynamic proxy).
➔For communicating with provider we need virtual object/proxy/port, service knows how to build port this is the service class provided by JAX-RPC not by generating the interface.

What is the difference between Stubs based consumer and Dynamic Proxy consumer?

Dynamic Proxy consumer

➔**Simple and fastest** way of building a consumer.
➔The Consumer we design is **portable** across all venders because we don't generating any binding classes using any vender provided tools.
➔When we work with DP no such **book keeping operation** is perform maintenance charge is very less.

Stub based consumer

➔Not simple to design /building a consumer.
➔The consumer we design is not portable across the vendor. Because we generate the binding classes using jax-rpc wscompile tool.
➔A huge maintenance cost required when we add any additional method/function in our provider side.
➔Because we have to always regenerate the binding classes when there an modification in provider side.
➔For this we have do "**book keeping operation** “means any changes in stub we have to keep a record manually which is a complicated job.

Q. Are you working in consumer? Have you ever get a chance of changing one vender to other?

➔Yaa!! I worked on consumer and of course we can change from one vender to other vender but I don't get a chance of working to change vender that is not there in our requirement. We just upgrade the version in our project.
➔If you want I can explain the procedure of changing one vender to other.

What you have to do when you work stub based consumer to change from one vender to other vender? Is there any alternative to avoid these messup?

➔**"Isolate the stub from the rest of classes with in your application"**
➔When our consumer communicates to the provider with the help of stub based consumer, in consumer side all the classes are depended to the stub/helper class.
➔Stub is communicated to the provider directly but other consumer classes are depended to stub for communicating.

➔If we changes one vender to other vender then we don't have to change all the classes of consumer, rather we change the **stub** because JAX-RPC provided a mechanism that our classes of consumer side not directly communicate to the stub.
➔They communicate through an **Invoker (Medium or layer)** that layer is communicating to the stub. In this way we have to change minimal code in our application.
➔There is a alternate to avoid these messup that is develop your consumer in Dynamic Proxy.

➔But In case of dynamic proxy zero code will be modify if we change one vender to other vender.

Why we don’t go for dynamic proxy based consumer always?

➔It’s not easy to debug the code in case of Dynamic proxy because there is no code with us and the code will generated at run time.
➔We communicate with provider by creating proxy. And we are not creating proxy with the help of tools. We have to write the logic in the **low level API** classes, which is complicated.
➔One who has good knowledge in Web services can write the logic in dynamic proxy.
➔It is difficult to develop the application using dynamic proxy comparing with stub based.
➔Performance will impact in case of dynamic proxy because proxy will create at run time.
➔To construct the proxy it takes time. But in case of stub based consumer the performance is high.

➔Technical point of view Stub based is easy but developer point of view DP is easy.

Simple Consumer Using Dynamic Proxy

WSDL Provided by Provider

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="BookInfoService"
                                  targetNamespace="http://bookinfoservice.org/sales/services"
                                  xmlns:tns="http://bookinfoservice.org/sales/services"
                                  xmlns="http://schemas.xmlsoap.org/wsdl/"
                                  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
                                  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/">
      <types/>

    <message name="BookInfo_getPrice">
         <part name="String_1" type="xsd:string"/>
    </message>
     <message name="BookInfo_getPriceResponse">
              <part name="result" type="xsd:float"/>
    </message>

     <portType name="BookInfo">
         <operation name="getPrice" parameterOrder="String_1">
              <input message="tns:BookInfo_getPrice"/>
              <output message="tns:BookInfo_getPriceResponse"/>
         </operation>
    </portType>

     <binding name="BookInfoBinding" type="tns:BookInfo">
         <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="rpc"/>
         <operation name="getPrice">
              <soap:operation soapAction=""/>
          <input>
                   <soap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
                                      use="encoded"
                                      namespace="http://bookinfoservice.org/sales/services"/>
           </input>
          <output>
              <soap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
                                      use="encoded"
                                      namespace="http://bookinfoservice.org/sales/services"/>
         </output>
         </operation>
    </binding>

    <service name="BookInfoService">
         <port name="BookInfoPort" binding="tns:BookInfoBinding">
              <soap:address location="REPLACE_WITH_ACTUAL_URL"/>
         </port>
    </service>

</definitions>
```

SEI Interface

```
package com.bis.services;

import java.rmi.Remote;
import java.rmi.RemoteException;

public interface BookInfo extends Remote {
      public float getPrice(String String_1)throws RemoteException;
}
```

```
public class DPTest {
        private static final String WSDL_LOC="http://localhost:8085/BookInfoService/bis?WSDL";
        private static final String SERVICE_NMSP="http://bookinfoservice.org/sales/services";
        public static void main(String[] args) throws MalformedURLException, ServiceException, RemoteException {
                ServiceFactory sFactory=ServiceFactory.newInstance();
                Service service=sFactory.createService(new URL(WSDL_LOC), new QName(SERVICE_NMSP, "BookInfoService"));
                BookInfo port=(BookInfo) service.getPort(new QName(SERVICE_NMSP, "BookInfoPort"),BookInfo.class);
                System.out.println(port.getPrice("isbn1008"));
        }
}
```

Consumer (Dynamic Proxy) complexType

WSDL Document

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

<definitions name="InsuranceProviderSev"
                                targetNamespace="http://www.icici.com/insurance/services"
                                xmlns:tns="http://www.icici.com/insurance/services"
                                xmlns="http://schemas.xmlsoap.org/wsdl/"
                                xmlns:xsd="http://www.w3.org/2001/XMLSchema"
                                xmlns:ns2="http://www.icici.com/insurance/types"
                                xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/">
      <types>
            <schema targetNamespace="http://www.icici.com/insurance/types"
                                xmlns:tns="http://www.icici.com/insurance/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="Member">
                  <sequence>
                    <element name="age" type="string"/>
                    <element name="gender" type="string"/>
                    <element name="memberName" type="string"/>
                    <element name="uniqueId" type="string"/>
                   </sequence>
          </complexType>
          <complexType name="Plan">
                  <sequence>
                    <element name="planName" type="string"/>
                    <element name="tenure" type="int"/>
                  </sequence>
                   </complexType>
          <complexType name="Policy">
                  <sequence>
                        <element name="memberId" type="string"/>
                        <element name="planName" type="string"/>
                        <element name="policyNo" type="int"/>
                        <element name="tenure" type="int"/>
                  </sequence>
                   </complexType>
            </schema>
      </types>

      <message name="Insurance_enroll">
         <part name="Member_1" type="ns2:Member"/>
         <part name="Plan_2" type="ns2:Plan"/>
      </message>

      <message name="Insurance_enrollResponse">
         <part name="result" type="ns2:Policy"/>
      </message>

      <portType name="Insurance">
         <operation name="enroll" parameterOrder="Member_1 Plan_2">
            <input message="tns:Insurance_enroll"/>
            <output message="tns:Insurance_enrollResponse"/>
            </operation>
      </portType>
      <binding name="InsuranceBinding" type="tns:Insurance">
            <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="rpc"/>
            <operation name="enroll">
                  <soap:operation soapAction=""/>
                  <input>
                  <soap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" use="encoded" namespace="http://www.icici.com/insurance/services"/>
                 </input>
                  <output>
                  <soap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" use="encoded" namespace="http://www.icici.com/insurance/services"/>
                 </output>
            </operation>
      </binding>
      <service name="InsuranceProviderService">
         <port name="InsurancePort" binding="tns:InsuranceBinding">
          <soap:address location="REPLACE_WITH_ACTUAL_URL"/>
         </port>
      </service>
</definitions>
```

Input Classes

```
package com.icici.services;
public class Member {
   protected java.lang.String age;
   protected java.lang.String gender;
   protected java.lang.String memberName;
   protected java.lang.String uniqueId;
   public Member() {
   }
   public Member(java.lang.String age, java.lang.String gender, java.lang.String
                                          memberName, java.lang.String uniqueId) {
      this.age = age;
      this.gender = gender;
      this.memberName = memberName;
      this.uniqueId = uniqueId;
   }
   public java.lang.String getAge() {
      return age;
   }
   public void setAge(java.lang.String age) {
      this.age = age;
   }
   public java.lang.String getGender() {
      return gender;
   }
   public void setGender(java.lang.String gender) {
      this.gender = gender;
   }
   public java.lang.String getMemberName() {
      return memberName;
   }
   public void setMemberName(java.lang.String memberName) {
      this.memberName = memberName;
   }
   public java.lang.String getUniqueId() {
      return uniqueId;
   }
   public void setUniqueId(java.lang.String uniqueId) {
      this.uniqueId = uniqueId;
   }
}
```

```
public class Plan {
   protected java.lang.String planName;
   protected int tenure;
   public Plan() {
   }
   public Plan(java.lang.String planName, int tenure) {
      this.planName = planName;
      this.tenure = tenure;
   }
   public java.lang.String getPlanName() {
      return planName;
   }
   public void setPlanName(java.lang.String planName) {
      this.planName = planName;
   }
   public int getTenure() {
      return tenure;
   }
   public void setTenure(int tenure) {
      this.tenure = tenure;
   }
}
```

Output Class

```
public class Policy {
   protected java.lang.String memberId;
   protected java.lang.String planName;
   protected int policyNo;
   protected int tenure;
   public Policy() {
   }
   public Policy(java.lang.String memberId, java.lang.String planName, int policyNo, int tenure) {
      this.memberId = memberId;
      this.planName = planName;
      this.policyNo = policyNo;
      this.tenure = tenure;
   }
   public java.lang.String getMemberId() {
      return memberId;
   }
   public void setMemberId(java.lang.String memberId) {
      this.memberId = memberId;
   }
   public java.lang.String getPlanName() {
      return planName;
   }
   public void setPlanName(java.lang.String planName) {
      this.planName = planName;
   }
   public int getPolicyNo() {
      return policyNo;
   }
   public void setPolicyNo(int policyNo) {
      this.policyNo = policyNo;
   }
   public int getTenure() {
      return tenure;
   }
   public void setTenure(int tenure) {
      this.tenure = tenure;
   }
}
```

Consumer Test class

```
public class TestDP {
        private static final String WSDL_LOCATION="http://localhost:8085/InsuranceProvider-LASTApproach/enroll?WSDL";
        private static final String SERVICE_NAME="InsuranceProviderService";
        private static final String URI_NS="http://www.icici.com/insurance/services";
        private static final String PORT_NAME="InsurancePort";

        public static void main(String[] args) throws ServiceException, MalformedURLException, RemoteException {

                ServiceFactory sFactory=ServiceFactory.newInstance();
                Service service=sFactory.createService(new URL(WSDL_LOCATION), new QName(URI_NS, SERVICE_NAME));
                Insurance insurance=(Insurance) service.getPort(new QName(URI_NS, PORT_NAME),Insurance.class);

                Member Member_1=new Member();
                Member_1.setUniqueId("1234 8765 0975 5673");
                Member_1.setMemberName("Dhananjaya");
                Member_1.setGender("Male");
                Member_1.setAge("27");

                Plan Plan_2=new Plan();
                Plan_2.setPlanName("Jivan Saral");
                Plan_2.setTenure(35);

                Policy policy=insurance.enroll(Member_1, Plan_2);

                System.out.println(policy.getPolicyNo());
        }
}
```

Dynamic Invocation Interface

➔It is similar to Java reflection technic.
➔In reflection we would be able to call a method on a class using the class name and method name dynamically at runtime on fly.
➔Similarly with Dynamic Invocation Interface, a client can call a remote procedure even the signature of the remote procedure or the name of the service is unknown until runtime.
➔In case of Stub or DP based client, we need to have SEI interface or generated classes to access the Provider.
➔But in case of DII we just need couple of things like Service Name, Port Name, Operation Name and Target Endpoint address etc to access the Service.

Steps to build a DII based client

```
public static final String SERVICE_NAME="BookInfoService";
public static final String PORT_NAME="BookInfoPort";
public static final String END_POINT="http://localhost:8085/BookInfoService/bis";
public static final String SERVICE_NAMESPACE="http://bookinfoservice.org/sales/services";
public static final String TYPE_NAMESPACE="http://www.w3.org/2001/XMLSchema";
```

➔Create the ServiceFactory –ServiceFactory is the class who knows how create the object of Service Interface.

```
ServiceFactory sFactory=ServiceFactory.newInstance();
```

➔Create the Service – We need to create the Service from the ServiceFactory. While create the Service and we need to give the input as Qualified Name of the Service From the WSDL Service element.

```
Service bookInfoService=sFactory.createService(new QName(SERVICE_NAMESPACE, SERVICE_NAME));
```

➔Create the call – Call is the Object which allows you to call a remote procedure on the Provider. To create the Call object, we need to call createCall() method on the Service Object. While creating the call object, to the createCall() method, we need to pass the Port Name from which you want to get the Call object.

```
Call getPrice=bookInfoService.createCall(new QName(SERVICE_NAMESPACE, PORT_NAME));

//Call getPrice=bookInfoService.createCall(new QName(SERVICE_NAMESPACE, PORT_NAME),new QName(SERVICE_NAMESPACE, "getPrice"));
```

➔Set the parameters on the Call object – The call just we retrieved is a generic Call object which is not pointing to any specific endpoint. Now we need to set the values like **method**

```
//Construct the Request
getPrice.setOperationName(new QName(SERVICE_NAMESPACE, "getPrice"));
getPrice.addParameter("String_1",new QName(TYPE_NAMESPACE, "string"),ParameterMode.IN);
getPrice.setReturnType(new QName(TYPE_NAMESPACE, "float"));
```

➔Along with that we need to set the below properties

```
//set the properties to request object
getPrice.setTargetEndpointAddress(END_POINT);
getPrice.setProperty(Call.ENCODINGSTYLE_URI_PROPERTY,"http://schemas.xmlsoap.org/soap/encoding/");
getPrice.setProperty(Call.SOAPACTION_URI_PROPERTY, "");
getPrice.setProperty(Call.SOAPACTION_USE_PROPERTY, true);
```

➔Invoke the method on the Call Object – To invoke the remote procedure on the service, we need to call Call.invoke(new Object[]{}) by passing required inputs as Object[] where it returns the response back to the user.

Consumer Using Dynamic Invocation Interface(SimpleType)

WSDL Document

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="BookInfoService"
                                          targetNamespace="http://bookinfoservice.org/sales/services"
                                          xmlns:tns="http://bookinfoservice.org/sales/services"
                                          xmlns="http://schemas.xmlsoap.org/wsdl/"
                                          xmlns:xsd="http://www.w3.org/2001/XMLSchema"
                                          xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/">
  <types/>

    <message name="BookInfo_getPrice">
        <part name="String_1" type="xsd:string"/>
    </message>
     <message name="BookInfo_getPriceResponse">
            <part name="result" type="xsd:float"/>
    </message>

     <portType name="BookInfo">
        <operation name="getPrice" parameterOrder="String_1">
            <input message="tns:BookInfo_getPrice"/>
            <output message="tns:BookInfo_getPriceResponse"/>
        </operation>
    </portType>

     <binding name="BookInfoBinding" type="tns:BookInfo">
        <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="rpc"/>
        <operation name="getPrice">
            <soap:operation soapAction=""/>
         <input>
                <soap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
                                 use="encoded"
                                 namespace="http://bookinfoservice.org/sales/services"/>
          </input>
         <output>
            <soap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
                                 use="encoded"
                                 namespace="http://bookinfoservice.org/sales/services"/>
        </output>
        </operation>
    </binding>

    <service name="BookInfoService">
        <port name="BookInfoPort" binding="tns:BookInfoBinding">
            <soap:address location="REPLACE_WITH_ACTUAL_URL"/>
        </port>
    </service>

</definitions>
```

Test Consumer

```java
package com.bis.services;

import java.rmi.RemoteException;

import javax.xml.namespace.QName;
import javax.xml.rpc.Call;
import javax.xml.rpc.ParameterMode;
import javax.xml.rpc.Service;
import javax.xml.rpc.ServiceException;
import javax.xml.rpc.ServiceFactory;

public class DiiTest {

        public static final String SERVICE_NAME="BookInfoService";
        public static final String PORT_NAME="BookInfoPort";
        public static final String END_POINT="http://localhost:8085/BookInfoService/bis";
        public static final String SERVICE_NAMESPACE="http://bookinfoservice.org/sales/services";
        public static final String TYPE_NAMESPACE="http://www.w3.org/2001/XMLSchema";

        public static void main(String[] args) throws ServiceException, RemoteException {
                //Construct the Call Object
                ServiceFactory sFactory=ServiceFactory.newInstance();
                Service bookInfoService=sFactory.createService(new QName(SERVICE_NAMESPACE, SERVICE_NAME));
                Call getPrice=bookInfoService.createCall(new QName(SERVICE_NAMESPACE, PORT_NAME));

                //Call getPrice=bookInfoService.createCall(new QName(SERVICE_NAMESPACE, PORT_NAME),new QName(SERVICE_NAMESPACE, "getPrice"));



                //Construct the Request
                getPrice.setOperationName(new QName(SERVICE_NAMESPACE, "getPrice"));
                getPrice.addParameter("String_1",new QName(TYPE_NAMESPACE, "string"),ParameterMode.IN);
                getPrice.setReturnType(new QName(TYPE_NAMESPACE, "float"));


                //set the properties to request object
                getPrice.setTargetEndpointAddress(END_POINT);
                getPrice.setProperty(Call.ENCODINGSTYLE_URI_PROPERTY,"http://schemas.xmlsoap.org/soap/encoding/");


                getPrice.setProperty(Call.SOAPACTION_URI_PROPERTY, "");
                getPrice.setProperty(Call.SOAPACTION_USE_PROPERTY, true);


                Float price=(Float)getPrice.invoke(new Object[]{"isbn1001"});
                System.out.println(price);


        }

}
```

Consumer (DII) complexType

WSDL Document

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

<definitions name="InsuranceProviderSev"
                                    targetNamespace="http://www.icici.com/insurance/services"
                                    xmlns:tns="http://www.icici.com/insurance/services"
                                    xmlns="http://schemas.xmlsoap.org/wsdl/"
                                    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
                                    xmlns:ns2="http://www.icici.com/insurance/types"
                                    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/">
       <types>
              <schema targetNamespace="http://www.icici.com/insurance/types"
                                    xmlns:tns="http://www.icici.com/insurance/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="Member">
                     <sequence>
                       <element name="age" type="string"/>
                       <element name="gender" type="string"/>
                       <element name="memberName" type="string"/>
                       <element name="uniqueId" type="string"/>
                      </sequence>
            </complexType>
            <complexType name="Plan">
                     <sequence>
                       <element name="planName" type="string"/>
                       <element name="tenure" type="int"/>
                     </sequence>
                      </complexType>
            <complexType name="Policy">
                     <sequence>
                              <element name="memberId" type="string"/>
                              <element name="planName" type="string"/>
                              <element name="policyNo" type="int"/>
                              <element name="tenure" type="int"/>
                     </sequence>
                      </complexType>
              </schema>
       </types>

       <message name="Insurance_enroll">
          <part name="Member_1" type="ns2:Member"/>
          <part name="Plan_2" type="ns2:Plan"/>
       </message>

       <message name="Insurance_enrollResponse">
          <part name="result" type="ns2:Policy"/>
       </message>

       <portType name="Insurance">
          <operation name="enroll" parameterOrder="Member_1 Plan_2">
              <input message="tns:Insurance_enroll"/>
              <output message="tns:Insurance_enrollResponse"/>
              </operation>
       </portType>
       <binding name="InsuranceBinding" type="tns:Insurance">
              <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="rpc"/>
              <operation name="enroll">
                     <soap:operation soapAction=""/>
                     <input>
                     <soap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" use="encoded" namespace="http://www.icici.com/insurance/services"/>
                    </input>
                     <output>
                     <soap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" use="encoded" namespace="http://www.icici.com/insurance/services"/>
                    </output>
              </operation>
       </binding>
       <service name="InsuranceProviderService">
          <port name="InsurancePort" binding="tns:InsuranceBinding">
            <soap:address location="REPLACE_WITH_ACTUAL_URL"/>
          </port>
       </service>
</definitions>
```

Test Consumer

```java
import java.rmi.RemoteException;

import javax.xml.namespace.QName;
import javax.xml.rpc.Call;
import javax.xml.rpc.ParameterMode;
import javax.xml.rpc.Service;
import javax.xml.rpc.ServiceException;
import javax.xml.rpc.ServiceFactory;

import com.icici.services.Member;
import com.icici.services.Plan;
import com.icici.services.Policy;

public class TestDII {
	private static final String ENDPOINT="http://localhost:8085/InsuranceProvider-LASTApproach/enroll";
	private static final String SERVICE_NAME="InsuranceProviderService";
	private static final String SERVICE_NS="http://www.icici.com/insurance/services";
	private static final String TYPE_NS="http://www.icici.com/insurance/types";
	private static final String PORT_NAME="InsurancePort";

	public static void main(String[] args) throws ServiceException, RemoteException {
		ServiceFactory sFactory=ServiceFactory.newInstance();
		Service service=sFactory.createService(new QName(SERVICE_NS,SERVICE_NAME));
		Call enroll=service.createCall(new QName(SERVICE_NS,PORT_NAME));


		//Define request
		enroll.setOperationName(new QName(SERVICE_NS,"enroll"));
		enroll.addParameter("Member_1",new QName(TYPE_NS,"Member"),ParameterMode.IN);
		enroll.addParameter("Plan_2", new QName(TYPE_NS,"Plan"),ParameterMode.IN);
		enroll.setReturnType(new QName(TYPE_NS,"Policy"),Policy.class);


		enroll.setTargetEndpointAddress(ENDPOINT);
		enroll.setProperty(Call.ENCODINGSTYLE_URI_PROPERTY,"http://schemas.xmlsoap.org/soap/encoding/");

		enroll.setProperty(Call.SOAPACTION_URI_PROPERTY,"");
		enroll.setProperty(Call.SOAPACTION_USE_PROPERTY, true);

		Member Member_1=new Member();
		Member_1.setUniqueId("1234 8765 0975 5673");
		Member_1.setMemberName("Dhananjaya");
		Member_1.setGender("Male");
		Member_1.setAge("27");

		Plan Plan_2=new Plan();
		Plan_2.setPlanName("Jivan Saral");
		Plan_2.setTenure(35);

		Policy policy=(Policy) enroll.invoke(new Object[]{Member_1,Plan_2});
		System.out.println(policy.getPolicyNo());
	}
}
```