WEB SERVICES

Distributed Application :-

1) In Java, we have standalone or distributed application.

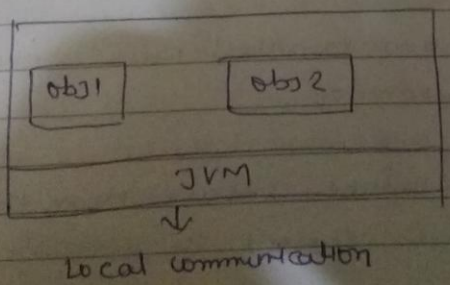
(2) A distributed application is a client/server application where a client and a server are running across network at various places.

(3) Distributed application in Java have two types.
   (I) Web based distributed application
   (II) Remote based distributed application

(4.) In Web based distributed application, a Browser will send a request to a server side object in a network

(5.) In remote based application, a Java object running at one JVM will send a request to another JVM object running at another JVM in network.

(6) In Java, two objects are running at one JVM then it is a local communication. and if two objects are running at different JVM then the communication is a distributed application.



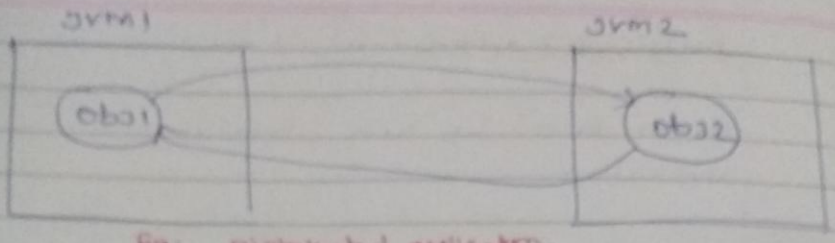


Fig: Distributed application.

(7) To develop remote communication application ie remote based distributed application we got first technology as socket programming in Java.

(8) In socket programming, the client application installed at various places will communicate with a server application across networks.

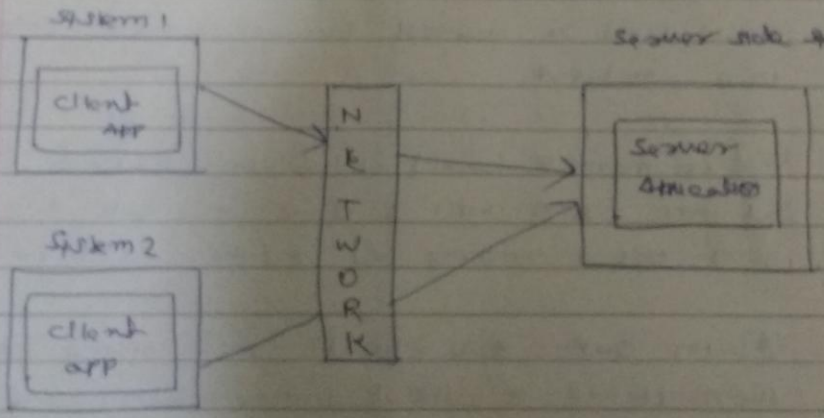


Fig: client / Server communication in socket programming

9) The major problem identified with socket programming is location transparency.

10) The local transparency problem mean if server application location is changed to another system

Naming registery solve the problem of location transparency.

then each client application can't connect with a server application.

(11.) To connect with a server application from client, a clients is need to be updated explicitly with new address of server application.

(12.) updating each client is a burden and a time consuming process. so, to resolve this location transparency problem, a Naming Registry is introduced in the middle of client and server.

Basic concept
Client-Server

The client is entity accessing the remote resource and the server provides access to server the resource. Operationally, the client is the caller and the server is the callee.

In Java term:

The client is the invoker of the method and the server is the object implementing the method.

The client and server can be homogeneous or heterogeneous: Some implementing language, different implementation language, some OS / different OS. Client and server refer both to the code and the system on which the web is running.

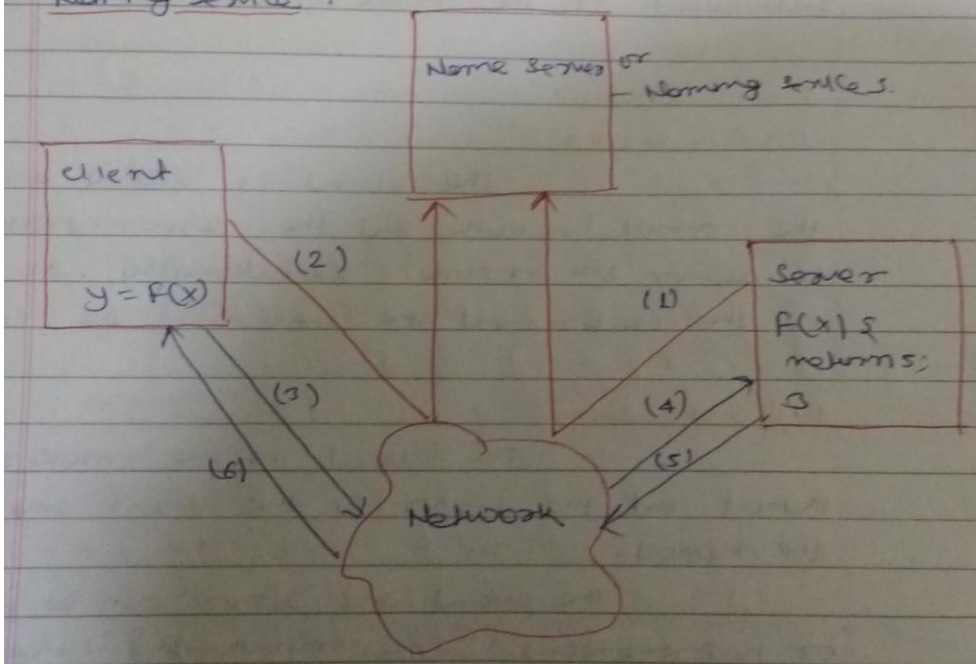
A distributed application may be homogeneous or heterogeneous.

for eg:

If client object and server object both are implemented in Java (Same language) then it is called distributed Homogeneous application.

If client object is in Java and server object is in other language then it is distributed Heterogeneous application.

In distributed application, location transparency problem is usually resolve with Name server or Naming service.



1) Register "F" with name server
(2) Lookup 'F' using name server

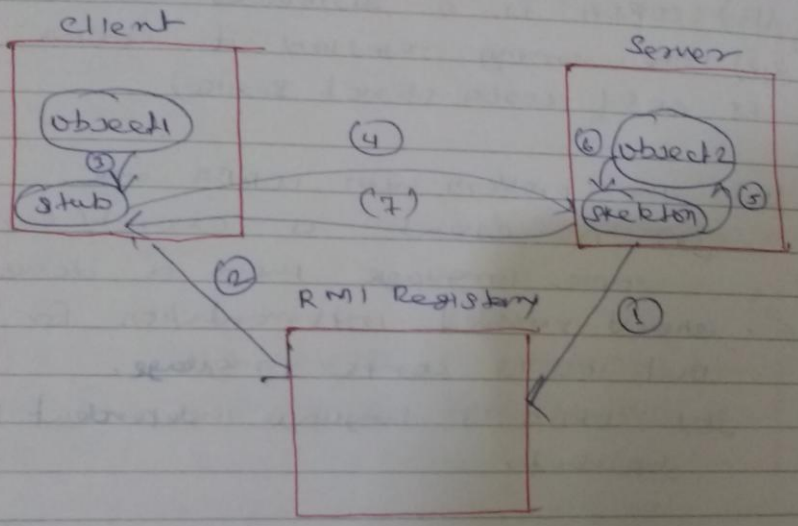
(3) Send message to call f with parameter
(4) Receive message that f was called with the parameter

(5) Send message with the result of calling f
(6) Receive message with the result of calling f.

Why Web Services are introduced?

(1) RMI:

(a.) RMI is a Sun technology with naming service.
(b.) In RMI, naming service is called RMI registry
(c.) RMI communication will be take like the following



The following major problems with RMI

(1) RMI is a distributed Homogeneous technology it means RMI can only communicate from Java to Java.

(2) RMI is a small API, so it is not suitable for constructing large distributed applications

(3) For communicating a client and Server in a network, RMI uses a protocol called JRMP

(Java Remote Method protocol). This protocol is not Firewall friendly.

(II) CORBA (Common Object Request broker architecture)

(I) CORBA is from OMG (object management group)
(II) CORBA introduced a neutral language called IDL (interface Definition language)
(III) CORBA is a distributed Heterogeneous technology
(IV) The naming registry of CORBA. CORDA is COS (corba object service)

(V) The problem with CORBA are,
(a) To create a client/server in some language then a developer should provided implementation for IDL interface But IDL is complex language.
(b) CORBA is language independent but platform dependent.

(VI) CORBA uses IIOP (internet inter ORB protocol) and it is a Firewall friendly

(III) EJB

(I) EJB is a distributed Homogeneous technology from Sun microsystems

(II) In EJB, the naming registry used b/w client and a server is JNDI registry (Java naming and directory interface)

(3.) When compared with RMI, EJB is firewall friendly because it uses IIOP protocol for communication.

(4) With EJB, we get the following problems

(a) EJB is a heavy weight technology. It requires EJB has a large specification and it is a complex technology to work.

(b) EJB can provide only Java to Java communication

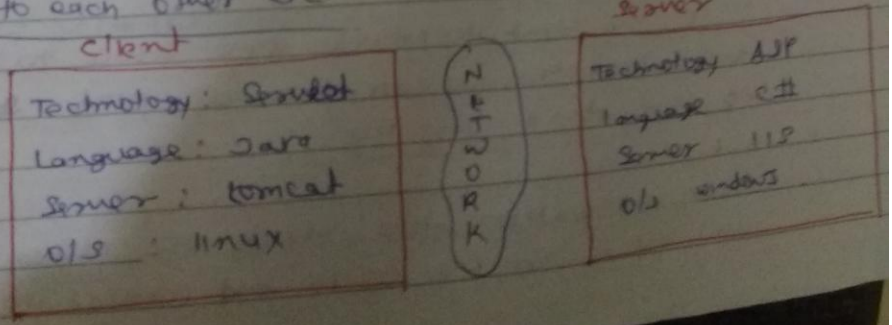
(c) Many classes and XML PDs are needed to develop an application.

Web Services :-

1) It is a distributed heterogeneous technology from W3C.

(2) The main purpose of WebServices technology is to provide interoperable communication b/w client and server.

(3.) The difference b/w previous technology and this WebServices is, it provides a client server interaction with language, technology, platform, server, browser etc, all these are independent to each other etc.



SOAP - Simple Object Access Protocol.

UDDI - Universal Description Discovery Interface.

WSDL - Web Service Description Language.

Q) What is difference b/w a Web application and a WebService?

Ans) Web application are created with intention to provide services to the client. This is also called customer to business interaction (C2B)

Whereas, Web Services are created with the intention of providing services to another application but not direct to the clients. This is also called business to business interaction (B2B).

A Web application also can act as client for Web Service.

client - which takes a service of Website. In CtoB C is customer for eg. and B2B, - Web application is client which it takes services to other web application.

If we have a online shopping Web application (Website) and a currency convertor Web service then a customer interacts with online shopping Website. and that Website interacts with Web service

Here, Website intention is to provide service to customers and Web-Service intention is to provide service to Website.

How a Service can be provided to other applications in Java:-

1) If we want to provide some service (logic) to other Java application then we can release that service in the form of a Jar file.

(2) An application who wants to utilize the service will add a Jar file to it and invokes the service.

interoperability means an application can access service which are written in any language & at any technology & any server.

(3.) A service is nothing but a method or a group of methods of a class to finish a particular task of application.

(4.) A problem of providing services in the form of Jar to application is, if a new version of service is created then an application has to get a new Jar file (if application wants take the new service) and it should be added to it to utilize the latest version of service.

(5) A Web service is also for providing service to other applications.

(6.) The difference b/w a providing a service in the form of Jar file and a Web service is, in case of Web service, the service will be available at server side and it is access by other application.

(7.) The benefits of web service in this scenario is, if any changes are made in service then it is automatically becomes available for all other applications.

(8.)

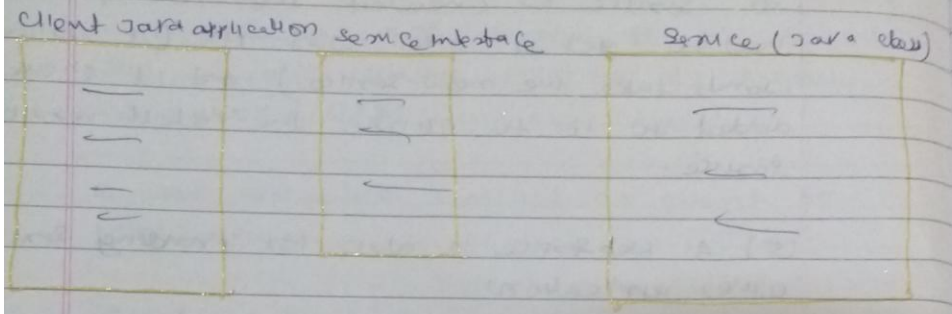
⇒ Important terminology of Web Services

(1) When communication across network is Java to Java then a Java service will inform to client about services offered in the form of

Jam

Java interface.

(2) The Java interface contains what services offered by a Java service class and what input value as parameter required by service and what output return by the service (method)



(3) In case of Web services, service can be one language and client may be some other language.

(4.) For example, if services created in Java and client is in c++ then a Java interface can't be understood by a c++ client

(5) So, in Web services communication, a common Mediator language is required b/w client and services of different language.

This common language for other language is XML.

(5) A WSDL file mainly contains the following three operation.

(i) Operations offered by a service
(ii) input needed to call a service.
(iii) Output produced by service.

(6) A WSDL file will be stored in a registory, to make it as available to the clients. This registory is called UDDI (Universal Descriptor discovery and integration)

(7) A client collects WSDL file from UDDI registory

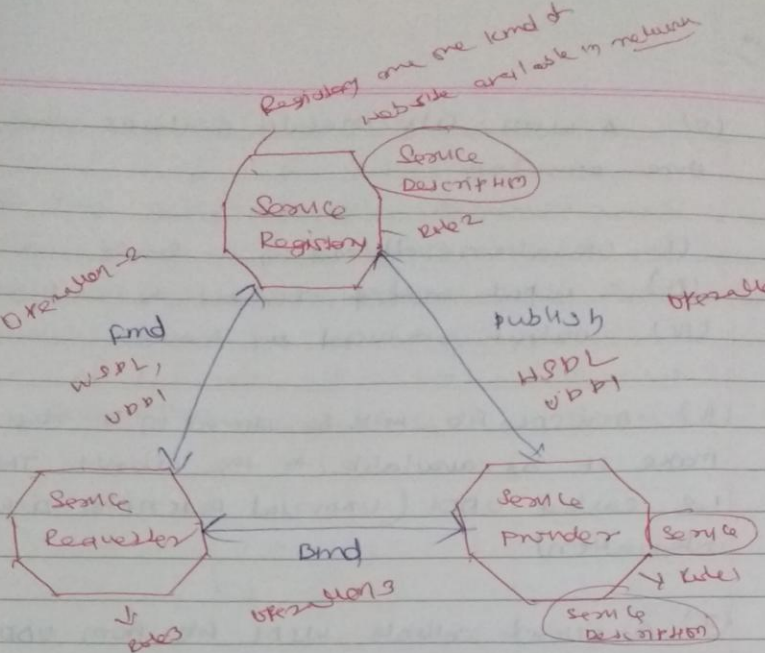
(8.) In Web services, a client and a service can be in different languages, so the datatypes of client and service application (at server side) do not matter.

(9.) Again, An XML format is used to call a service and also to get a response of a service. This XML format is called SOAP (simple object access protocol.

Web Services Architecture:

Web services architecture (SOA) has two components.

(i) Roles
(ii) operations.



Description about above frame.

Roles:

(i) Service provider:-

(a) Service provider is an entity who define a web service

(b) A service provider do the following:

(i) create a description of service

(ii) deploy the service in runtime environment (Web/app server) to make it easy to access across network

(iii) publishes the service-description to service registry

(2) Service Registry

(1) A service registry is to enable to match-making between service provider and service requestor.

(2) once match was found then interactions are carried out directly b/w a service requestor and a service provider

(3) A service registry will do the following.

(a) Accepts request from service providers to publish and advertise Web service description.

(b) Allows service requestors to search for service description contained within service registry.

(3) Service Requestor

A service requestor is a component which consume WebService across network.

A Service Requestor do the following task

(i) Finds a service description published in service Registry.

(ii) uses the service description to bind and invoke the WebService hosted by service provider.

(I) operations

(i) publish

(ii) find

(iii) bind

(i) publish :

① The publish operation performs service registration or advertisement of service.

② When a service provider publishes the its web services in a registory then it is advertising the service to all requestors in a network.

(ii) Find :-

A find operation performs the operation of service searching a service based on some condition. The result is a service description that matches the search criteria.

(iii) Bind :-

The Bind operation enables a relationship b/w client / server for service requestor and service provider.

Extensible Markup Language (XML) :-

(1.) XML is a technology from W3C

(2) XML and HTML are siblings of SGML (standard generalize Markup language)

(3.) The one technology we use in java projects, which is in non-java programming XML.

(4) Diff b/w HTML and XML are given

a.) HTML is to display the data and XML is to describe a data.

(b) HTML is case-insensitive and XML is a case-sensitive.

(c.) HTML file contains all predefine tags.
XML file contains all user-defined tags.

Note :-

1) <h1> 123 </h1> → Here H1 display the data in Heading 1.

(2) <bookid> 123 </bookid> → Here bookid describe the data.

Two Types of XML Documents :-

(1) Well-formed xml document.
(2) Valid xml document

(1) To say that An XML document is well-formed, the document should follow the below rules.

Rule

(a) The document should contain a root element.

(b.) each start tag should contains an end tag.

(c.) If a parent tag followed by a child

tag is opened then while closing first child tag followed by parent tag should be closed.

(4.) The attributes values should be quoted.

(5.) A program or application which checks the well formedness and valid ness of XML documents is called XML parser.

(6) We have two kinds of parser
(i) non-validating parser

(I) Validating parser.

(I) If an XML parser checks only well formedness then it is non-validating parser.

(II) If an XML parser checks for both well formedness and validness then it is called validating parser

(III) eg.
each web browser has an XML parser and it checks well formedness so it is a non-validating parser.

(IV) eg.
Each Web container has an XML parser and it checks for both well formedness and validness. so it is a validating parser

9.) What is valid XML documents?

(1) If XML document is created by following the structure defined by vendor then it is called a valid xml document.

(2.)

(3) The structure of XML document will be given by a vendor in the formed of a DTD (document Type definition) or XSD (XML schema definition) file.

eg. SUN Microsystems has given a DTD file for constructing web.xml. So a developer should construct web.xml according to the DTD.

8) The following DTD file defines structure that root tag is a persons and it can have one or more person tags.

<!Element persons (person+)>

<!Element person (pid, pname, mobile#)>

<!Element pid (#PCDATA)>

<!Element pname (#PCDATA)>

<!ELEMENT pmobile (#PCDATA)>

Here
+ → for 1 or more times

* * ⇒ for 0 or more times

* +! ⇒ DataType indicator

* PCDATA ⇒ Parse character Data, it is data type.

In addition to

* ? ⇒ for 0 or one time

* If there is no symbol then exactly one time occurrence will be shown.

⇒ A DTD file can be imported into an XML file by using a <!DOCTYPE> tag name

Syntax

<!DOCTYPE> <rootelement> PUBLIC "-//
owner of root element dtd// purpose of
dtd // language code, "URI">

eg:

Sathy persons.xml

<!DOCTYPE persons PUBLIC "-// Sathy
Technology// "c:/sekhar/xml/persons.dtd">
person info dtd// EN
URI

<persons>
<person>
<id> 1234 </id>
<name> Dee </name>
<mobile> 1284 </mobile>

</person>

</persons>

XML Schema Definition

(1.) Schema is another way of defining the structure of xml document and it is advanced to DTD

(2) With DTD, the following problems are identified

(i) A DTD has given three symbols.

(a) *

(b) +

(c) ?

To specify the number of occurrences of an element But ~~DTD~~ XSD has not given any minimum and maximum occurrence specifiers.

eg If we want to specify that an element should occur for minimum 3 times and maximum 5 times then we don't have any symbol for this

(ii) DTD has very less data types and there is no way given to create user-defined data type.

(iii) DTD increases burden on xml parser because a parser has to follow different rules to check the dtd and different rules to check the xml is valid or not.

(4) To overcome the above problems of dtd, W3C has introduced Schema as an alternate for DTD

(5) The structure of XML document will be defined in file folder XSD file (XML schema definition).

(6) TO define the structure in XSD file, we use tags and datatype given by W3C and namespace

Namespace :-

A namespace is a collection of names (tag name and datatype name).

We can think a namespace as equal to a Java package.

- A Java package contains a collection of classes and XML namespace contains a collection of names.

A namespace identified by some URI and it may be a valid or invalid URI.

We need to think namespace URI as some id (identifier).

For constructing an XSD file, whatever the tags and datatypes required are given by W3C under the namespace URI "http://www.w3.org/2001/XMLSchema".

XML Schema

Namespace:-

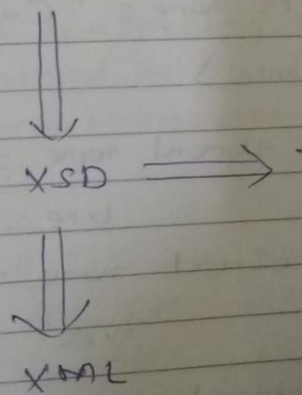
While creating an xsd file, the tags and datatypes required for constructing xsd are given by W3C in a namespace called

http://www.w3.org/2001/XMLSchema.

While creating an XSD file, new tags and new datatypes are generated. The newly generated tags and datatypes will be stored in another namespace called target namespace.

While creating an XML file, target namespace will be imported into XML file and then tags of target namespace are used to constructing an XML document.

eg. http://www.w3c.org/2001/XMLSchema → W3C given namespace



To used Tags and DataTypes of namespace in either XML file or XSD file then namespace must be imported with the help of keyword called xmlns.

customers.xsd

<xsd:xmlns:>

<xsd:schema xmlns:xsd="http://www.w3.org/2001/xmlschema" targetNamespace="http://sathya.org/2004">

<xsd:element name="customers" type="CustomerType" />

<xsd:complexType name="customerType">

- <xsd:element name="customer" type="customerType" minOccur=1 maxOccur=3 /> → user define

</xsd:complexType>

<xsd:complexType name="CustomerType">

<xsd:sequence>

<xsd:element name="customerid" type="xsd:int" />

<xsd:element name="customername" type="xsd:string" />

<xsd:element name="phone" Type="xsd:long" /> → predefined

</xsd:sequence>

</xsd:complexType>

</xsd:schema>

Some points on customer.xsd file

In the above xsd file we used tags, schema, element, complexType and sequence. We also used datatype int, long and string. These are all given by W3C under it's namespace called http://www.w3c.org/2001/XMLSchema. This is sometimes space.

In the above xsd file. We created new tags customers, customer, customerid phone, customername and datatypes, customerType and customerTypes. These are all stored in target space namespace: "http://sathya.org/1234",

⇒ XSD file contains only single namespace given by W3C but in xml file, only targetname space will be imported.

⇒ While constructing xml documents, to use tags, defined by in a XSD file, we need to import targetnamespace used in XSD file into our xml file.

⇒ In xml file we need to attach the location of XSD file also with the tag <SchemaLocation> given by W3C under XmlSchema-instance namespace so we need to import this namespace into our xml file.

⇒ Schema location contains two parts for each name space both parts are separated with a whitespace.

customer.xml

<customers xmlns="http://sathya.org/2014"

2) xmlns:xsi="http://www.w3.org/2001/xmlschema-instance

3) xsi:schemaLocation="http://sathya.org/2014 http://sathya.org/2014/customers.xsd">

<customer>

<customerId> 123 </customerId>

<customerName> ABC </customerName>

<phone> 12345 </phone>

</customer>

</customers>

1) TO

XML parser :-

An XML parser is a program (class), it checks for Wellformedness and Validness of xml documents.

In Java we have two validating parsers called as DOM (Document Object Module) and SAX (Simple API for XML).

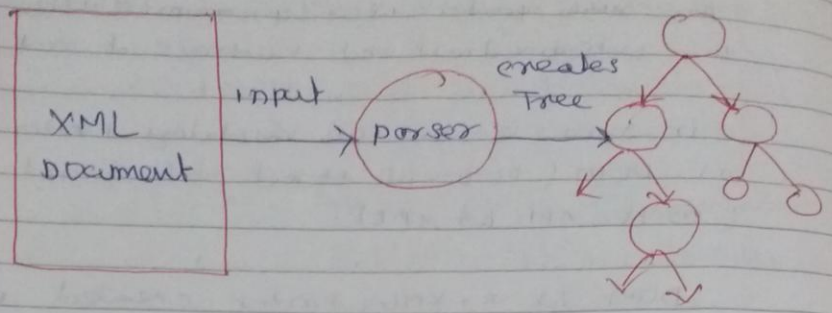
DOM is a XML parser created using specification given by W3C.

SAX parser is own parser of Java.

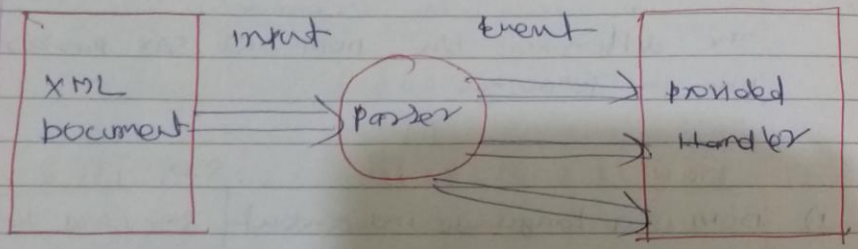
The difference b/w DOM and SAX parser will be like the following.

DOM	SAX parser
DOM is a language independent parser. It means DOM parser is existing for multiple language.	Sax is a language dependent parser it means SAX parser exists only in JAVA
DOM is a Tree based parser. It means DOM parser create a tree structure for xml document. If it is valid	SAX is a Event Driven parser. It means SAX parser fires events.
DOM is a Heavy Weight parser	SAX is a Light Weight parser and fast parser
DOM parser can be used to read the content of XML document and also write the content into XML.	(4) SAX parser can be used only for reading the content of XML

DOM parser



SAX parser



Simple object Access protocol (SOAP) :-

1) Soap is a specification given by W3C and its current version 1.2

2) Soap is not a protocol. it is a kind of XML document (message).

(3.) SOAP is introduced to transfer input and output b/w a service requestor and service provider in the form of XML

Message.

In Web Services communication a client and server participating in communication may be in same language or may be in different language so a common mediator format is needed to transfer the data and that is SOAP.

- SOAP messages are transfer b/w a client and a server by inserting them into http request and http response.

So SOAP Messages are firewall friendly.

In Web application communication the request and response, the document which contains a header part and body part.

Both of these parts are sparated with a blank line.

If a request is sent through Get Method then the body part of request document will be empty and for Post Method the body contains input value.

A Sample request document with GET and POST Method will be look like the following

request (document)

(Header part)
GET /srvl?uname=sathya&pwd=Java HTTP
User-Agent: mozila firefox4.0
Accept Language :- en-us
contact-length : 200
(blank line)
(body part) <<empty>>

request (document)

POST /srvl? ~~uname=sathya~~ HTTP/1.1 (Header part)
User-Agent ; mozila firefox4.0
Accept language : en-us
contact-length : 200
(blank line)
uname = sathya & pwd = Java (body part)

In Web Services communication, The soap request message and SOAP response message are xml message format which are constructed internally by some tools.
The SOAP request and response mess XML message are constructed by using the tag given by W3C under a namespace

http://schemas.xmlsoap.org/soap/envelope

A Soap message contains three tags

(1) <Soap: Envelope>

prefix (may be any name)

(2) <Soap: Header>

(3) <Soap: Body>

in a SOAP request and response message, Header part is optional and Body part contains either input or output.

A SOAP message is also, XML message and it follows XML rules eg of SOAP document

<SOAP: Envelop xmlns: soap = "http://schemas.xmlsoap.org/soap/envelope/">

<soap: Header />

<soap: Body>

</soap: Body>

</soap: Envelop>

Sample for SOAP Request:

```
<SOap: Envelope xmlns:soap2 " http://schemas.
                xmlsoap.org/soap/envelope/">
  <soap: Header />
    <soap: Body>
      <ns2: Hello xmlns:ns2 = "http://
            sahya.javo.org/sample.
        <arg0> Web Services </arg0>
      </ns2: hello>
    </soap: Body>
</soap: Envelope>
```

Sample for SOAP Response

```
<soap: Envelope xmlns:soap = "http://schemas.
                xmlsoap.org/soap/envelope/">
  <soap: Header />
  <soap: Body>
    <ns2: helloResponse xmlns:ns2 = "http://
          sahya.java.org/sample">
      <return> Hello Web Services </return>
```

</ns2:helloresponse>

</soap:Body>

</soap:Envelope>

WEB SERVICE DESCRIPTION LANGUAGE (WSDL):-

⇒ WSDL is a kind of XML document.

⇒ In WEB-SERVICES, A service client and service provider both may be in some language and different language. So, To establish communication between service client and provider, WSDL is generated.

⇒ We can say that WSDL establish a contract between a service requester and service provider.

⇒ While creating a web service or a client for WebService, there are more tools support for generating WSDL or for parsing (reading) WSDL.

⇒ A Web service creator or a service client only need the knowledge of WSDL, to better understand WebService communication.

The following are the seven element of WSDL File.

(a) <Definitions> → root tag of WSDL
(b) <types>
(c.) <message>
(d) <operation> <port type>
(e.) <Binding> <operation>
(f) <binding>
<service>

(1.) <Definitions>

(2.) <types> :-

⇒ <Types> element is used for importing the target namespace in which the xml datatypes of webservices operation are stored.

⇒ Before creating WSDL document, internally XSD file created and in that XSD file two xml datatypes or elements are created for each operation and they are stored target namespace.

⇒ Using this types tags, the XSD created internally and target namespaces are imported

(3) <message>

1) This message tag is for will convert xml types into WSDL message type.

2) In Web Services communication, in the body part of SOAP documents, an input and output should be send in the form of WSDL message only.

3) For Each operation of WebService, Two messages are created where one for input (request) and other for response (output).

(4) <PortType> & <Operation>

A portType is a abstract specification of a WebService's operation.

A port type is like an Interface of java because an interface is an abstract specification of methods

A portType has one or more operation and each operation of input and output tags.

A WebService's client can understand an operation of WebService with the help of <portType>

(5) <Binding>

This tag assigns a Transport protocol to be used for communication and SOAP message style used for communication

The SOAP message style can be Document/literal or RPC/literal

(7) <Service>

A Service is a collection of ports.

A port number can be attach, URL of Web Service address to be binding.
of WebService

The URL indicate the exact network location where a WebService is running.

A Sample of WSDL document

<Definition targetNamespace = "http://org.sathya.com
name = "DemoWebService.Service" >

<types>

<xsd:schema>

<xsd:import namespace = "http://org.sathya.com
schemaLocation = "http://localhost:2222/
SampleWebApp/DemoWebService?xsd=1" />

</xsd:schema>

</types>

<message name = "hello"

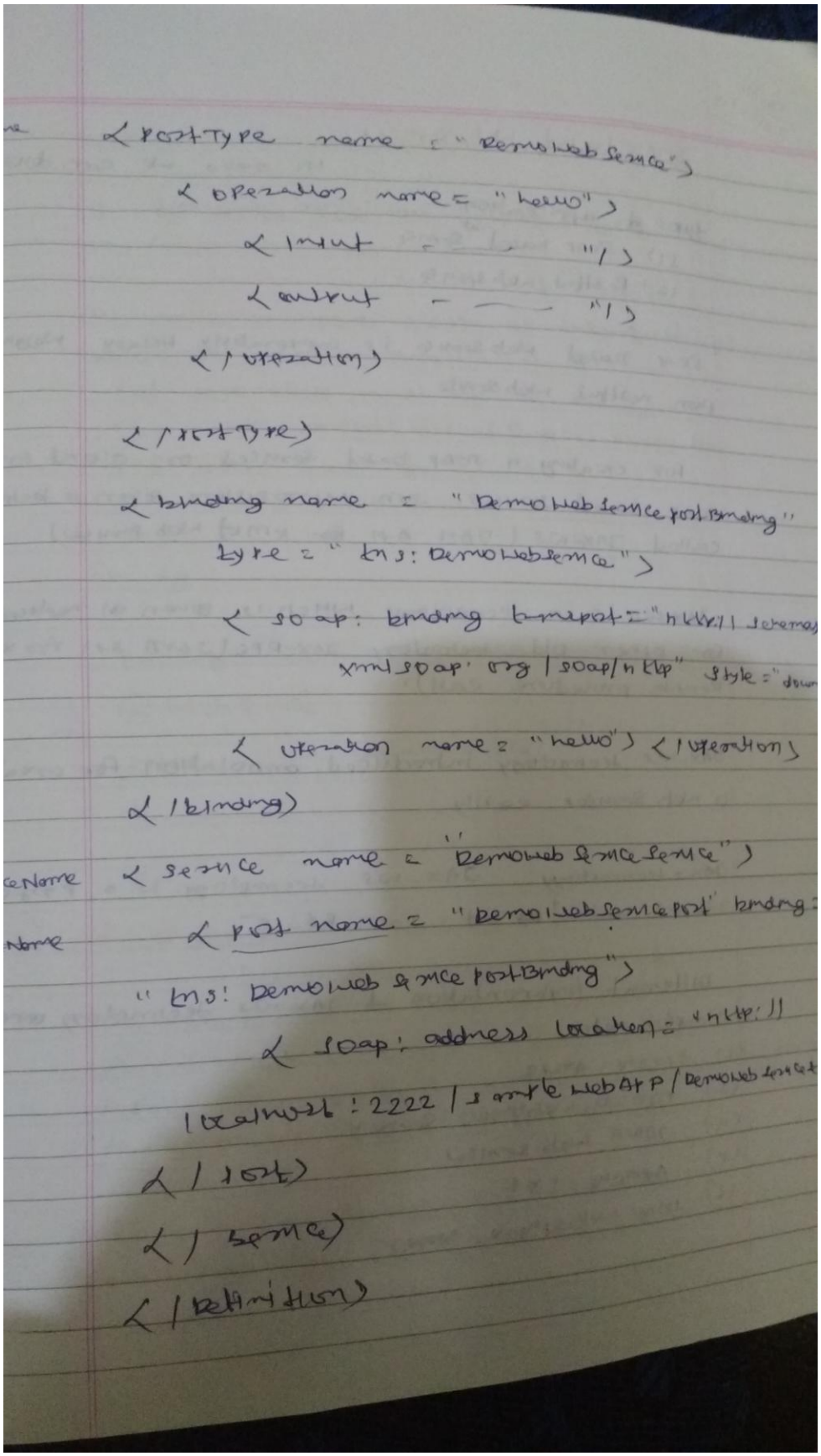
<part name = "parameter" element = "tns:hello" />

</message>

<message name = "helloResponse" >

<part name = parameter element = "tns:hello
Response" />

</message>



SOAP based Web Service

In java we can develop two types of application.

(1) SOAP based web service
(2) Restful web service.

SOAP based web service is comparatively heavy weight than restful web service.

For creating a SOAP based services or client for SOAP based services, sun microsystem given a technology called JAX-WS (JAVA API for XML Web services).

JAX WS is a technology which is given as replacement for other old technology JAX-RPC (JAVA API for Remote procedure call).

JAX-WS technology introduced annotation for creating a web service easily.

JAX-WS technology is a part of JDK 1.6 update and Java EE 5.

Different implementation of JAX-WS technology are

(1) Glassfish metro
(2) Apache AXIS2
(3) Oracle Weblogic Server
(4) JBoss Web service
(5) Apache CXF
(6) IBM Websphere server

Creating a WebService:-

A Web Service can be created either by the following POJI / POJO or POJO model.

TO make a POJO class as Web Service, it should follow the below rules.

(a) POJO class must be in package.
(b) We should @WebService annotation.
(c) class must be public
(d) class must contain a default constructor.

for eg.

```
package com.sathyg.webservices.soap;
@WebService
public class CatalogService
{
=
}
```

(I) @WebService annotation:

The parameters of this annotation are,

(1) serviceName:-

(a) This parameter is used to customize the service name to be used in wsdl document.

(b) If the parameter is not given then by default the service name will be created as Web Service classname + Service.

For eg

@WebService (serviceName = "cs")

Public class catalogService
{ }

→ In WSDL document, the service tag will be
<service name = "cs" >

eg2
@WebService, (without any parameter)

Public class catalogService
{ }

→ In wsdl document, the service tag will be
<service name = " catalogServiceService" >

(2) name :- (portType)

This parameter is used to customize portType name to be used in WSDL document.

If class implements WebService class interface then by default interface name will be used as port type name.

If a web service class doesn't implements any interface then class name is used as port type name.

eg

@WebService
public class CatalogService
{ }

In wsdl document, portType name will be
<portType name = "CatalogService" >

eg:

@WebService (name = "CatsService")

public class CatalogService

{
}

In wsdl document, portType name will be
< portType name = "CatsService" >

(3) targetNamespace → (package Name will be used as targetNamespace in reverse order)

This parameter is used to specify a URI for XML Type (WSDL XSD) for WebService operation.

URI specified may be a resolver resolvable or may not resolvable. If parameter are not passed for all @WebService annotation then package name in reverse order will be taken as targetNamespace.

(4) PortName :-
This parameter is used to specify the portName to be used in WSDL document.

If Name parameter is passed for the @WebService annotation then that name + Port will be taken as default as portName.

If portName parameter is not passed then ClassName (WebService class) + Port will be taken as default portName.

If we want to customize, the port name then we passed this port name parameter

eg

①

```
@WebService(name = "cs")
public class catalogService
{
}
```

Here portname in WSDL file will be
<port name = "csPort">

②

```
@WebService
public class catalogService
{
}
```

→ Here, port name in WSDL file will be

<name = portName = "catalogServicePort">

eg

(3) @WebService(portName = "myCatalogPort")

public class CatalogService
{
}

→ Here portname in wsdl file will be
<port name = "myCatalogPort">

(5) endpointInterface:

This parameter is to specified fully qualified interface name which is implemented by a WebService class.

This parameter is mandatory if the class or WebService implement interface. otherwise it is optional.

eg. @WebService(endpointInterface = "com.sujya.
WebServices.soap.catalog")

public class CatalogServices implements catalog

@WebMethod annotation:

This annotation is optional on top of the method and it has two parameter.

(a) operationName
(b) exclude

If operationname parameter is not passed then by default method name becomes of operation

name in wsdl document.

If we want to exclude a method as operation from a wsdl document then we pass exclude = true (default value is false)

for eg

@ WebMethod (operationName = "Hello")
public string sayHello()
{
}

⇒ Here, the operation name in wsdl file will be <operation name = "Hello">

eg
@ WebMethod
public string sayHello()
{}

Here the operation name in wsdl file will be <Operation name = "sayHello">

@ WebParam annotation :

This annotation is applicable or added for parameters of "web service method.

If this annotation is not added for a parameter of a WebService method then in soap request message, the tag is used of

<arg0> <arg1> etc.

If we want to change this args tag with some parameter name then we use this @WebParam annotation with its parameter as called name.

eg

public string SayHello (string txt)
{
}

⇒ In SOAP request, <soap: Body> tag contains, <arg0> tag.

eg2

public string SayHello (@WebParam (name = "text") string txt)
{
}

In SOAP request, <soap: Body> tag contains, <text> tag.

@WebResult annotation

This annotation is used to change the (return) tag in SOAP response to some other tag.

This annotation is applicable only for method with return type other than void.

This annotation is applicable on top of web service method.

eg:

```
public string sayHello(string txt)
{
    return "Hello : " + txt;
}
```

In SOAP response, <soap:Body> tag contains <return> tag.

eg2

```
@WebResult(name = "message")
public string sayHello(string txt)
{
    return "Hello : " + txt;
}
```

In SOAP Response, <soap:Body> tag contains <message> tag.

Example:

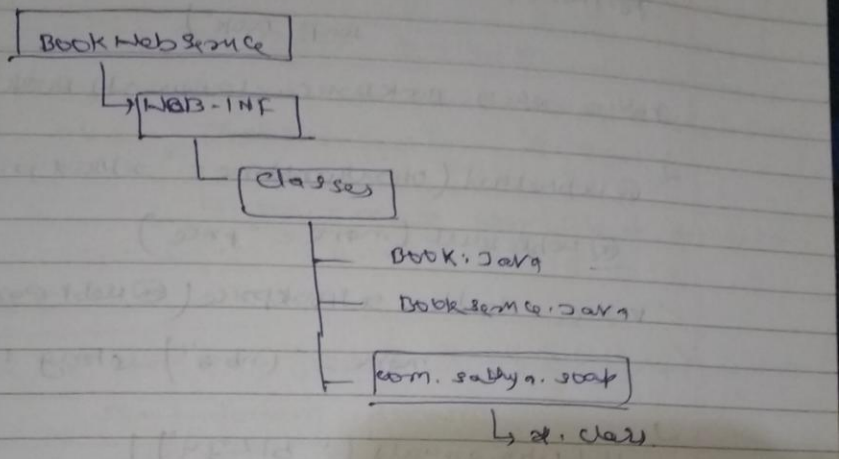
- This first application contains a WebService in POJO/POJI model with one operation called: getBookPrice.
- This getBookPrice operation takes input as ISBN number and returns price of that book.

TO make this WebService as accessible for all the clients across the network, we are storing this WebService on top of a Web application and we are deploying the application in

Creating Server.

Step 1

1) create a directory structure for the web application.



(2) create Book.java and BookService.java

```
//Book.java

package com.sathya.soap;
import javax.jws.WebService;
public interface Book
{
    double getBookPrice(String isbn);
}
```

```
// BookService.java

package com.sathya.soap
import javax.jws.WebService;
import javax.jws.WebMethod;
import javax.jws.WebResult;
```

```
import javax.jws.WebParam;

@WebService(serviceName = "BookService", name = "BookPortType
service" targetNamespace = "http://N.example.subya.org"
portName = "BookPort", endpointInterface = "com.subya.
                soap.Book")
public class BookService implements Book
{
  @WebMethod(operationName = "getBookPrice")
    @WebResult(name = "price")
    public double getBookPrice(@WebParam(
                name = "isbn") String isbn)
  {
    if(isbn.equals("51234"))
        return 590.99
    else if(isbn.equals("65432"))
        return 458.99;
    else
        return 0;
  }
};
```

steps

compile the same code with package compilation

Step 6 create a War file for the application.
create a War file from root folder.

```
C:\> Application_WebService\BookWebService\WEB-INF\classes> javac -d . *.java
```

↳ For package compilation.

```
C:\> WebApplication_WebService\BookWebService> jar cvf BookService.war *.*
```

↳ To create War file of application.

Step 7

```
C:\Program Files\Glassfish 4.0\glassfish\bin\asadmin start-domain domain1
```

Step-domain

Open the cmd mode with administrator then execute above cmd.

⇒ Start Glassfish Server.

⇒ With NetBeans IDE, we will automatically get glassfish server and we can use it directly.

⇒ To start the glassfish server, type the following at command prompt which are written above inside box.

Steps

Open admin console page of Glassfish server on browser like the following.

http://localhost:4848

ster

① at left side select Application ⇒ click on deploye button ⇒ click on browse button ⇒ select BookService.war ⇒ open ⇒ OK

Note

To check whether our webService is successfully deployeed or not, open wsdl file on browser using the following url.

http://localhost:4455/BookService/BookService?wsdl

↓ war file name ↓ Service Name

Creating a console Application as client for above

⇒ create a folder with name as client in c drive

⇒ Type wsdlimport command at command prompt like the following

c:\client> wsimport -p book1 -keep

↓ package Name ↓ to keep the source code

http://localhost:4455/BookService/BookService?wsdl ↵

⇒ Define BookClient.java like the following and save it under client folder

```
import book1.*;
class BookClient
{ p.s.v.m(String[] args) throws Exception
  {
    BookService service = new BookService();
    Book book = service.getBookPort();
    double price = book.getBookPrice("b1234");
    sop("price of b1234 is: "+price);
  }
}
```

compile and execute

O/P price of b1234 is: 590.99

⇒ Steps to Develop a Web Service with NetBeans 7.3

Steps

1.) File ⇒ New project ⇒ choose Java Web at left side ⇒ Web Application at right side ⇒ Next ⇒ give the project name ⇒ next ⇒ Next ⇒ Finish.

(2.) expand Web pages at left side and delete index.jsp.

(3.) Right click on Application Name (App2) ⇒ New ⇒ other ⇒ select Web Service from left side ⇒ select Web Service at right side ⇒ Next ⇒ Enter ⇒ Web Service Name (SampleService) ⇒ Enter package Name (com.sathya.soap) ⇒ Finish

eg:
```
package com.sathya.soap;
import javax.jws.WebService;
import javax.jws.WebMethod;
import javax.jws.WebParam;

@WebService(serviceName = "SampleService", targetNamespace =
                                                    "http://soap.sathya.com/")
public class

Public class SampleService {
    @WebMethod(operationName = "hello")
    public String hello(@WebParam(name = "name") String
                                                    txt) {
        return "Hello" + txt + " !";
    }
}
```

(4) Right click App2 ⇒ Redeploy

(5) To test a Web Service, expand Web Services Node at left side ⇒ Right click on SampleService ⇒ click on TestWebService ⇒ enter some input in text box ⇒ click on Hello hello button

(6) in order to see the WSDL document, type the following Url at address bar

http://localhost:4455/App2/SampleService?Wsdl

creating a console application of Java as client for the above WebService using netbeans NetBeans 7.3

To create client

Step

(1) File ⇒ New ⇒ Java at left side ⇒ JavaApplication at right side ⇒ Enter ProjectName client1 ⇒ uncheck create a Main class check box ⇒ Finish

(2) Right click on client1 ⇒ New ⇒ Other ⇒ WebServices at left side ⇒ Web Service Client at right side ⇒ select WSDL Url and enter Url as http://localhost:4455/App2/SampleService?Wsdl ⇒ Finish

(3) copy the package org.sathya.example from generated some folder into source package folder

(4) Right click on source package ⇒ New ⇒ Java class ⇒ Enter class name as Test ⇒ Finish

```
public class Test {
    public static void main(String args[])
    {
        String str = hello("Java");
        sop (str);
    }
    private static String hello(java.lang.String
                                name) {
        org.sathya.examples.SampleService service
        service = new org.sathya.examples.SampleService();
        org.sathya.examples.SampleServicePort port =
        service.getSampleServicePort();
        return port.hello(name)
    }
}
```

Note:-

(1) In the above Test class, main method is calling hello() method of Test class and that hello() method is calling hello() of WebService.

(2) The local hello() of Test class can be automatically renamed like the following

expand Web Service not at right side ⇒ expand sample service ⇒ expand sample service port ⇒ drag hello() onto Test class.

⇒ creating a console application as a client for above Web Service without IDE

~~create a folder~~

Step

1) create a folder with the name client in c drive.

(2.) Open a command prompt and type the following ~~wsdl~~ wsimport command for generating client code in java for a wsdl file.

c:\client > wsimport -p pack1 -keep http://localhost:4455/App2/sampleService?wsdl.

o/p

parsing WSDL

generating code

compiling code

-p → which specify package name

-keep → to keep the source code in package.

(3) create a class with public static void main like the following and save it under client

Folder.

```
import pack1.*;
class Test
{
    public static void Main(String args[]) throws
                                        Exception
    {
        SampleService_Service service = new SampleService
                                        _Service();

        SampleService ss = service.getSampleServicePort();
        String str = ss.hello("Abcd");
            System.out.println(str);

    }
}
```

compile and run

Javac Test.java

Run - Java Test.

Example3

creating another Web service from NetBeans 7.3.

=> We are creating a WebService with the name calculator Interface.

The calculator service class contains one operation called getCalculate(), it will take input as calculated and returns a JavaBean object.

1) File ⇒ New ⇒ project ⇒ Java Web ⇒ Web App ⇒ next ⇒ projectName (App) ⇒ next ⇒ next ⇒ Finish

2.) expand Web pages at left side and delete index.jsp.

(3.) Right click on projectname ⇒ new ⇒ other ⇒ select Java at left side ⇒ Java interface at right side ⇒ next ⇒ enter class name (catalog) and package name is com.sathya.soap ⇒ finish

```
Package import javax.jws.WebService;
Package com.sathya.soap;
@WebService
Public interface Catalog
{
    List getCatalog(String catalogId);
}
```

4.) Right click on App ⇒ new ⇒ Other ⇒ Select Web Service at left side ⇒ Web Service at Right side ⇒ next ⇒ enter Web Service name as Catalog Service and package name com.sathya.soap ⇒ Finish.

// CatalogService.java

```
package com.sathya.soap;
import javax.jws.WebService;
import javax.jws.WebMethod;
import javax.jws.WebParam;
```

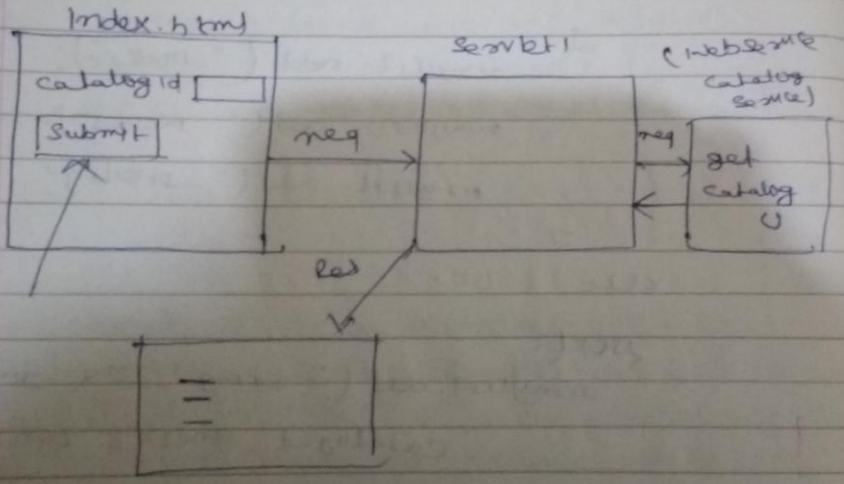
```
import java.util.List;
import java.util.ArrayList;

@WebService(serviceName = "CatalogService", endpoint
        interface = "com.sathya.soap.catalog")
public class CatalogService implements Catalog
{
    public List getCatalog(String catalogId)
    {
        ArrayList arrayList = new ArrayList();
        if (catalogId.equals("COOT"))
        {
            arrayList.add("books");
            arrayList.add("movies");
            arrayList.add("mobs");
        }
        else
        {
            arrayList.add("Sorry! The given input
                catalogId doesn't exist")
        }
        return arrayList;
    }
}
```

Right click on projectName (Alt+2) ⇒ deploy.

Calling a Web Service from a Servlet

⇒ We are creating one HTML page and we are submitting the request to a Servlet.

⇒ From Servlet we are calling a Web Service called CatalogService by passing catalogId as input



1) File Menu ⇒ New project ⇒ Java Web ⇒ Web Application ⇒ Next ⇒ Enter projectName (CatalogClientApp) ⇒ Next ⇒ Next ⇒ Finish.

(2) Delete index.jsp file from Web Pages folder

(3) Write click on project name ⇒ New ⇒ Other ⇒ Select WebService from left side of WebService client at right side ⇒ next ⇒ select WSDL URL (enter like http://localhost:4455/ABC/catalog?wsdl) ⇒ enter package name (optional) Packl ⇒ finish

(4.) Right click on project name ⇒ New ⇒ HTML ⇒ Enter filename as index.html. ⇒ finish.

```
// index.html
<body>
    <form action="jsp1">
        catalog id: <input type="text"
        name="t1"> <br>
        <input type="submit" value="submit">
    </form>
</body>
```

(5.) Right click on project name ⇒ new ⇒ Servlet ⇒ Enter class name as Servlet1 ⇒ next ⇒ select Add information to deployment descriptor ⇒ Enter Url pattern as /jsp1 ⇒ finish

```
protected void processRequest(HttpServletRequest
        request, HttpServletResponse response)
{
    PrintWriter out = response.getWriter();
    String catalog2 = request.getParameter("t1");
```

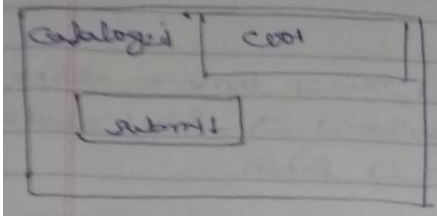
```
List list = getCatalog(catalogId);
Iterator it = list.iterator();
while(it.hasNext())
{
    Object o = it.next();
    out.println(o);
    out.println("<br>");
}
out.close();
}
private java.util.List getCatalog(java.lang.
                    String arg0)
{
    packt.Catalog port = service.getCatalog
                    ServicePort();
    return port.getCatalog(arg0);
}
}
```

Note

In the above code, we will get the method getCatalog() automatically like the following.

At leftside expand Web Service References → expand Catalog Service → Catalog Service → Catalog Service port → getCatalog drag getCatalog method into Servlet.

⇒ Right click on catalog clientApp ⇒ run



iv) Steps to Add Glassfish Server to the eclipse id

(1.) Start eclipse id and enter some workspace name.

(2) Window menu ⇒ show view ⇒ Servers ⇒ in Servers view click on new Server wizard ⇒ click on download additional server adapters ⇒ Select Glassfish tools ⇒ next ⇒ Next ⇒ Finish.

(3) After restarting the ide, GO to Server view ⇒ click on new Server wizard ⇒ select glassfish 3.1 ⇒ next ⇒ click on browse button and select the path C:\Program Files\glassfish-3.1.2\glassfish ⇒ next ⇒ password is optional ⇒ Finish

(4) TO Test Whether a glassfish server is successfully started or not ⇒ Right click on glassfish server view ⇒ ⇒ view admin console

If we are calling a web service and exception is occur in web service then what happens

contract first service last? JAXB

service first contract last! JAXB

⇒ Creating a Web Service in Eclipse

Step

1) click on file Menu ⇒ new ⇒ Dynamic Web project ⇒ Enter project Name ⇒ WebService1 ⇒ Select Glassfish 3.1 as target run time ⇒ finish

(2) Right click on project name ⇒ new ⇒ class ⇒ Enter package name (com.sathya.soap) and class name DemoWebService ⇒ finish.

```
// DemoWebService

package com.sathya.soap;
import javax.jws.*;
@WebService(serviceName="DemoWebSP", name=
        "DemoWebService", portName=
        "DemoWebport", targetNamespace="http://
        examples.sathya.soap");
public class DemoWebService

{
    @WebMethod(operationName="sayHello")
    public String sayHello(String username)
    {
        return "Hello :"+username;
```

Step 3

Right click on project Name ⇒ Run as ⇒ Run on Server ⇒ Select glassfish server ⇒ finish.

http://localhost:4455/WebService1/DemoWeb.asmx?wsdl

To check a Web Service is successfully deployed or not type the given above url at address bar.

Testing the above Web Service by without creating any client application in .Net

Step 1

(i) Click on Launch Web Service Explorer icon.
(ii) Click on WSDL page at right side.
(iii) Click on WSDL Main link at left side.
(iv) Enter WSDL url like http://localhost:4455/WebService1/DemoWeb.asmx?wsdl and click Go

Step 2

(i) Click on SayHello (it is operation name)
(ii) Click on Add link and enter input value in a text box (like Sathya) and click on Go

(iii) The output of Web Service operation can be seen in status box at bottom.

14

Creating a console app^n as a client for eclipse Webservice using Eclipse IDE →

Step1

File → new → Java project → enter project name as client → next → finish

Step2

Right click on project name → new → other → expand Web Services → select "WebServiceClient" → next → enter Service definition as : "http://localhost:4455/WebServ/DemoWeb?wsdl" → output folder "/client/src" → select ☐ define customize mapping for namespace to package → next → add

namespace : "http://examples.sathya.soap"

package : pack1 → finish

Step3

At left side expand src → right click on pack1 → new → class → enter name : "TestClient" → select public static void main checkbox → finish

```
// TestClient.java
package pack1;
public class TestClient
{
    p s v m (String[] args) throws Exception
    {
        DemoWebServiceProxy proxy = new DemoWebServ
        DemoWebService service = proxy.getDemoWebService
        String output = service.sayHello("ABCD");
```

```
    sop(output);
  }
}
```

right click on TestClient → run as → Java Application.

eg

The following example contains one operation on Web Services, it takes input as item object and returns output as item class object.

The Web Service operation connects with database, reads the details of an item based on ItemId and returns the details in the form of item object.

```
// Item.java
package com.sathya.soap;
public class Item
{
    private int itemId;
    private String itemName;
    private double price;

    public int getItemId() {
        return itemId
    }
```

```
//OrderWebService.java
Package com.sathya.soap;
import javax.jws.WebService;


@WebService(serviceName = "OrderWebService")
public class OrderWebService
{
    public Item getItemDetails(Item item)
    {
        Item i = new Item();
      try
      {
        class.forName("oracle.jdbc.OracleDriver");
    Connection con = DriverManager.getConnection
                ("jdbc:oracle:thin:@localhost:
                    1521:XE", "system", "manager");
    preparestatement pstmt = con.prepareStatement
                ("select * from item where itemId=?
    ResultSet rs = pstmt.executeUpdate();
    pstmt.setInt(1, item.getItemId());
    ResultSet rs = pstmt.executeQuery();
      if(rs.next())
      {     i.setItemId(rs.getInt(1));
```

```
i.setItemName(rs.getString(2));
i.setPrice(rs.getDouble(3));
}
rs.close();
rstmt.close();
    con.close();
}
catch(Exception e)
{
    System.out.println(e);
}
return i;
}
}
```

⇒ Add ojdbc14.jar to the lib folder of the project

⇒ Deploy the project in server ⇒ Run as ⇒ Run on server

⇒ Create a table with the name items and insert sample rows into the table.

SQL> create table items(itemid number(5) primary key, itemname varchar2(10), price number(9,2));

insert row into table

⇒ insert into items values(102, 'samsung', 9000);

commit.

⇒ creating a console application as a client for the above web service.

Step

1) File → New ⇒ Java project ⇒ New ⇒ Enter project Name ⇒ (clientorder) ⇒ Finish

2) Right click on project Name ⇒ New ⇒ Other ⇒ expand Web Services ⇒ Web Service client ⇒ Next ⇒ Enter service definition http://localhost:4411/orderproject/orderwebservice?wsdl ⇒ Next ⇒ Finish.

3) Right click on src folder ⇒ New ⇒ class ⇒ Enter Name (: Client) select public static void Main ⇒ Finish.

```
// client.java
public class client
{
    public static void Main(String[] args)
                    throw Exception.
    {
    OrderWebServiceProxy proxy = new
            OrderWebServiceProxy();
    OrderWebService_PortType port = proxy.getOrder

    Item it = new Item();
        it.setItemId(101);
```

```
Item item = port.getItemDetails(10);
sop("Item id=" + item.getItemDetails().getId());
sop("Item name=" + item.getItemDetails().getName());
sop("Item price=" + item.getPrice());
```

⇒ Right click on Runas ⇒ Java application.

O/P

Item id = 101
Item name = sony
Item price = 8500.00

Creating a WebService in ~~top~~down approach in eclipse :-

Step1 file ⇒ New ⇒ dynamic Web project ⇒ enter project name as "TOP DOWN Test ⇒ Finish.

2. Right click on project (TOP DOWN Test) ⇒ new ⇒ Other ⇒ expand WebServices ⇒ select "WSDL file" ⇒ Next ⇒ enter file name as "demo.wsdl" ⇒ next ⇒ enter target namespace as "http://example.satya.soap ⇒ prefix as "tns" ⇒ select wsdl skelton "check box ⇒ protocol = soap ⇒ soap binding = document literal ⇒ finish.

Steps: select "design" tab of Demo.wsdl ⇒ double click on new operation word and change operation name as sayHello and press enter ⇒ save it.

step4.

Step 4

Right click on project name ⇒ new ⇒ Other ⇒ expand Web Services ⇒ select "Web Service" ⇒ next ⇒ select "Web Service type" as "top down Java bean Web service" ⇒ click on browse ⇒ again click on browse ⇒ select wsdl file ⇒ ok ⇒ next ⇒ enter skeleton folder as "/TopDown/src" ⇒ finish

Step 5

In demo.soap.impl.Java file enter the following return statement under sayHello() as

```
public String sayHello(String rn)
{
    return "Hello World";
}
```

Step 6

Right click on project ⇒ run As ⇒ run on server.

Now, creating a console application as a client for the above web service:-

Step

1) file ⇒ New ⇒ Java project ⇒ next ⇒ enter project name as "client of TopDown ⇒ finish.

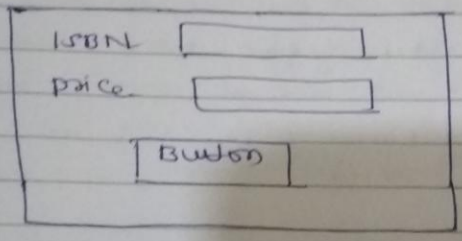
2.) Right click on project name ⇒ new ⇒ other ⇒ expand "Web Services" ⇒ select "Web Service client" ⇒ next ⇒ browse ⇒ browse ⇒ select "demo.wsdl" ⇒ ok ⇒ ok ⇒ next ⇒ enter output folder as "/client of TopDown/src ⇒ finish

Creating a C#.net app^{n} as a client for BookService application of Java.

(1.) Start ⇒ programs ⇒ Microsoft Visual Studio 2101.

(2.) File ⇒ new ⇒ project ⇒ select Visual C# ⇒ Windows forms App^{n} ⇒ enter name as "FormApp" ⇒ OK.

(3.) Design the form like the following :-



(4.) click on "project" menu → "add ServiceReference" enter address as "http://localhost:4455/BookService/BookService.svc?wsdl" ⇒ go ⇒ enter namespace as "ServiceReference1" ⇒ OK

(5.) double click on the button and enter the following code in Button_click() as :-

```
{
    ServiceReference1.BookClient client = new
        ServiceReference1.BookClient();
string input = textBox1.text
    double output = client.getbookprice(input);
    textBox2.Text = output.ToString();

}
```

soap12 support enabled

(5) "Debug" Menu ⇒ Start debugging ↵

⇒ Creating a Web Service in ASP.Net

(1.) Start ⇒ Programs ⇒ Visual Studio 2010 ↵
(2.) File ⇒ New ⇒ project ⇒ Visual c# at left and ASP.Net Web appl^n at right side → enter name as "Web Service Appl" ⇒ OK ↵

(3.) Click on "project" menu ⇒ add new item ⇒ select "Web Service" ⇒ add.

(4) Enter the following code under "Web Service1" class.

```
public class WebService : System.Web.Services.WebService
{
    [WebMethod]
    public string HelloWorld()
    {
        return "JAI RAM ji ki";
    }
    [WebMethod]
    public string SayHello(string str)
    {
        return "Hello -- " + str + "!";
    }
    [WebMethod]
    public string SayBye()
    {
        return "Bye!";
    }
}
```

(5.) "debug" menu ⇒ start debugging.

(6) To see the wsdl file, type the following in address bar:-
"http://localhost:1292/WebService1.asmx?wsdl"

⇒ Creating a console application of a client for the above WebService of .Net.

Step

1) create a project folder with the name "client".

(2.) Type wsimport command at command prompt like:→

C:\client> wsimport . -p pack1 -keep http://localhost:1292/WebService1.asmx?wsdl ↵

parsing wsdl...
Generating code...
Compiling code...

(3.) create a client program like the following and save it under client folder.

```
import pack1.WebService1;
import pack1.WebService1Soap;
class client
{
    psvm(String[] args) throws Exception
    {
        WebService1 service = new WebService1();
        WebService1Soap wsp = service.getWebService1Soap();
```

```
sopln ( soap. helloWorld ());
sopln ( soap. sayHello ("Ram"));
sopln ( soap. sayBye ());

}
```

3

step4: e:\ javac client.java

e:\ java client ↵

```
// JaxbXmlToJava.java
package com.sathya.jaxb;
import java.io.File;
import java.util.ArrayList;
import javax.xml.bind.*;
public class JAXBXMLToJava {
    public static void main(String[] args) {
        try {
            // create JAXB context and initializing
            Unmarshaller
            JAXBContext jaxbContext = JAXBContext.newInstance
                                        (Country.class);

            Unmarshaller jaxbUnmarshaller = jaxbContext.
            createUnmarshaller();

            // specify the location and xml file to be read
            File xmlFile = new File("C:\\Sathya\\CountryFeed.xml");

            // this will create Java object - country from the
              xml file
            Country countryInJava = (Country) jaxbUnmarshaller.
                unmarshal(xmlFile);
            System.out.println("CountryName: " + countryInJava.getCountryName());
            System.out.println("CountryPopulation: " + countryInJava.getCountry
                Name() + population());

            ArrayList<State>
            stateList = countryInJava.getStateList();
```

```
        int i=0;
        // for each loop
        for(State state : stateList)
        {
            i++;
            System.out.println("State: " + i + " " +
                state.getStateName());
        }
    } catch (JAXBException e) {
        // some exception occured
        e.printStackTrace();
    }
}
}
```

O/P

~~output~~

JAXBExample.java com.sathya.jaxb.JAXBXMLToJava

O/P

CountryName :

CountryPopulation : 50000000

State 1 : AP

State 2 : MP

Apache Axis2 :-

⇒ axis → apache extensible interaction system

⇒ By default Apache Tomcat doesn't contain a WebService engine, in order to process WSDL file for the deployed Web services and creating for the SOAP documents needed.

Apache introduced axis as Web Service engine, to support Webservices deployed in a Tomcat server.

⇒ Apache axis is a WebServices engine which supports soap based services.

⇒ As a successor of axis, apache introduced Axis2 as a WebService engine to supporting soap based and restful services.

⇒ Before we deploye a WebService in Tomcat. First of all we need to add the Axis2 Web Services engine to Tomcat Server.

⇒ Axis2 WebServices can be added to the following Tomcat like the following

(a) Visit http://axis.apache.org/axis2/java/core/download.cgi

(b) Download War distribution zip file of version of 1.6.2 version

(c) Extract the axis2 War zip file downloaded

then we get axis2.war file

copy axis2.war into tomcat/webapps folder

TO Test axis2.war is added successfully or not, then start the server and Type the following request

http://localhost:2014/axis2/

Date 16/05/14

Developing a Web Service for axis2

Actually WebService is a class and make it as accessible over the network. A Web Services class must be a part of Web app.

If we want to run a WebService in a Server with axis2 WebServices engine then the Web application will be provided by axis2 Web Services engine only. Just we need to copy our WebService in the form of aar (apache axis2 archive) file to the axis2 Web application.

TO create a Web Service and to prepare aar file to the WebServices class we need to follow a folder structure.

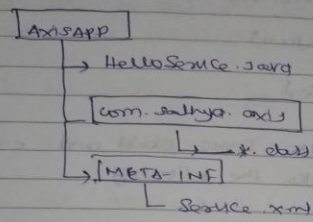
(i) create a folder to place WebService class

(ii) create a sub folder with name META-INF and store services.xml under the META-INF sub folder

The following Web Service is a Hello Service with one operation called sayHello.

Step

1) create a directory like the following



// HelloService.java

```
package com.sathya.axis;
public class HelloService
{
    public string sayHello()
    {
        return "Hello";
    }
};
```

(2) // <!-- Services.xml -->

```
<service>
    <parameter
        name = "ServiceClass" > com.sathya.axis.
        HelloService </parameter>
    <operation name = "sayHello" >
        <MessageReceiver
            class = "org.apache.axis2.rpc.
            receivers.RPCMessageReceiver" />
    </operation>
</service>
```

(3) compile the service class and create aar file like the following

a) Javac -d . *.java
b) Jar cvf Hello.aar .
c) copy Hello.aar file into Tomcat6/webapps/axis2/WEB-INF/services folder
d) Start the Tomcat6 Server
e) To see wsdl file of our Web Service, open the browser and type the following address

localhost:2014/axis2/

after it click on Services ⇒ click on Hello

⇒ Developping a client application for Calling axis2 Web Service.

In we use wsdl2Java tools to prepare client side file needed for a wsdl file generated by axis2 web service

There tools wsdl2Java and jar files needed

to compile the generated .java code for wsdl file, we need to download axis2 binary distribution zip file (axis2-bin-.zip) and then extract it.

We need to set the following three environment variables for wsdl2java tools.

(1) Set path = %path%; D:\axis2-1.6.2\bin

(2) Set JAVA_HOME = C:\Program Files\Java\jdk1.7.0

(3) Set AXIS2_HOME = D:\axis2-1.6.2

Step

1.) Create a folder with name axisclient in some drive

2.) Type the following wsdl2java tools at command prompt

C:\axisclient> wsdl2java -p pack1 -uri http://localhost:2014/axis2/services/Hello?wsdl

(3.) The above wsdl java tools create a folder with name src and create package under src i.e pack1 and stored generated java file.

(4.) Create a client application and save it under src like the following

```
// Client.java
import pack1.*;
class client
{
public static void main(String[] args)
{
HelloStub stub = new HelloStub();
HelloStub.SayHello obj1 = new HelloStub.SayHello();
HelloStub.SayHelloResponse obj2 = stub.SayHello(obj1);
String str = obj2.get_return();
System.out.println(str);
}
}
```

In HelloStub class, for each web service method two nested classes one created where one class object is used for sending the input and another class object is to store the output.

In our Web service class, we have a service method SayHello() so two nested classes created in HelloStub one SayHello and SayHelloResponse

In order to read the return value of stub a web service method stored in response object we call get_return method.

TO compile the java file generated by wsdl2java tool under the package.
TO compile java file generated by the tools we need to set the following

In classpath

1) axiom-api-1.2.13 (D:\axis2-1.6.2\lib)
(2.) axiom-impl-1.2.13.jar
(3.) axis2-adb-1.6.2.jar
(4.) axis2-kernel-1.6.2.jar
(5.) axis2-transport-http-1.6.2.jar
(6.) axis2-transport-local-1.6.2.jar
(7.) commons-codec-1.3.jar
(8.) commons-httpclient-3.1.jar
(9.) commons-logging-1.1.1.jar
(10.) httpcore-4.0.jar
(11.) mail-1.4.jar
12.) neethi-3.0.2.jar
13.) wsdl4j-1.6.2.jar
14.) XmlSchema-1.4.7.jar

goto src → pack1 → javac ~~classes~~ *.java

cd..

src) javac client.java

java client

Hello

⇒ Create a client application using for the Book Service deployed in glassfish server

Step

1) Start the glassfish server

2) Create a folder "BookAccessClient" under e drive

(3) Fire the following command at command prompt

(4) C:\BookAxisClient> wsdl2java -uri http://localhost:4455/BookService/BookService?wsdl

(4.) ~~import~~ import soap.entity.example

class client

{

public static void main (String[] args) throws Exception

{

BookServiceStub stub = new BookServiceStub();

BookServiceStub.GetBookPrice obj1 = new BookServiceStub.GetBookPrice();

obj1.setArg0("b1234");

BookServiceStub.GetBookPriceE obj1e = new BookServiceStub.GetBookPriceE();

obj1e.setGetBookPrice(obj1);

BookServiceStub.GetBookPriceResponseE obj2e = stub.getBookPrice(obj1e);

BookServiceStub.GetBookPriceResponse obj2 = obj2e.getGetBookPriceResponse();

```
double price = obj2.getreturn();
    System.out.println("price ="+ price);
  }
}
```

(or) C:\Book AxisClient\src\sathya soap\sathya\examples\ javac *.java

C:\Book AxisClient\src\java client

price = 530.33

⇒ Developing Axis2 webservice in eclipse :-

1) Click on Window menu ⇒ preferences ⇒ expand web service at left side ⇒ select Axis2 preferences ⇒ Axis2 preference ⇒ Browse ⇒ select axis2-1.6.2 folder ⇒ OK ⇒ OK

2) Add Tomcat server to IDE

(3.) Click on File menu ⇒ Dynamic Webproject ⇒ Enter project Name (HelloAxis2) ⇒ select Dynamic web module version as 2.5 ⇒ click on modify button ⇒ select Axis2 webservices checkbox ⇒ Finish.

(4) Right click on src ⇒ new ⇒ class ⇒ Enter package name ⇒ (com.sathya.axis2) ⇒ Enter name HelloService ⇒ Finish ⇒

```
package com.sathya.axis2;
public class HelloService
{
    public String sayHello(String name)
    {
        return "Hello -- "+ name;
    }
}
```

(5) Right click on project ⇒ new ⇒ expand web services ⇒ select webservice ⇒ next

(6) click on browse button and select HelloService as serviceimplementation ⇒ click on web service runtime and select Apache Axis2 and click on next ⇒ select generated default services.xml file ⇒ next ⇒ next ⇒ start server ⇒ next ⇒ finish

Note

To test whether a web service successfully deployed or not type the following url to see wsdl file.

http://localhost:2014/HelloAxis2/services/HelloService?wsdl

⇒ creating a client application for the above service in eclipse

(1) Click on File Menu ⇒ new ⇒ other ⇒ expand web service ⇒ select web service client ⇒ next ⇒ browse button of url (http://localhost:2014/HelloAxis2/services/HelloService?wsdl

(2) Select web service runtime Apache axis2

(3) Select client project ⇒ enter client project name (clientservice) ⇒ client project type (dynamic web project) ⇒ next ⇒ finish

(4) go to src of client project ⇒ Right click on package of project name ⇒ new ⇒ class (client) ⇒ select main ⇒ finish.

```
package com.satya.axis2;
public class client {
public static void main (String args[]) throws
Exception
{
HelloServiceStub stub = new HelloServiceStub();
HelloServiceStub.SayHello obj1 = new HelloServiceStub.SayHello();
obj1.setName("ABC");
HelloServiceStub.SayHelloResponse obj2 = stub.sayHello(obj1);
System.out.println(obj2.get_return());
```

Right click on client ⇒ run as ⇒ java application

Code Satya

Accessing a currency convertor web service of WEBSERVICEX.NET by creating a client application in netbeans.

Step

1) Open www.webservicex.net website ⇒ click on currency convertor at left side of website ⇒ find the WSDL url of currency convertor web service

(2) start netbeans ide ⇒ File Menu ⇒ New project ⇒ Java Web ⇒ web application ⇒ enter project name (currency client) ⇒ next ⇒ finish Next ⇒ project finish

(3) Right click on currency client ⇒ new ⇒ other ⇒ select Web Services at left side ⇒ Web Service client at right side ⇒ next ⇒ select WSDL url and enter collected from currency convertor web service (http://www.webservicex.net/currencyconvertor.asmx?WSDL) ⇒ enter package (pack1) ⇒ finish

(4) Remove the existing default code of index.jsp

(5) expand currency client ⇒ expand web service reference ⇒ expand currency convertor ⇒ expand currencyconvertor soap ⇒ drag and drop convertible into index.jsp.

```
// index.jsp
<%-- start web service invocation --%>
```

```
<!--
try
{
    pack1.CurrencyConvertor service = new pack1.CurrencyConvertor();
pack1.currencyConvertorSoap port = service.getCurrencyConvertorSoap();
pack1.Currency fromCurrency = null; pack1.Currency.USD;
pack1.Currency toCurrency = null; pack1.Currency.INR;
double result = port.conversionRate(fromCurrency, toCurrency)
out.println("Result = " + result);
} catch (Exception e)
{
}
-->
```

(6.) Right click on CurrencyClient ⇒ Run

Creating a client in a netbeans, for the US Weather WebService of a third party cdyne.com.

Step

File ⇒ New project ⇒ Java Web ⇒ Web App ⇒ provide name ⇒ (Weather Client) ⇒ Next ⇒ Next ⇒ finish

2) Right click on project name → new → Web Service client → wsdl → http://www.cdyne.com/Weather/Weather.asmx?wsdl → package (pack1)

⇒ Answer

(a) Remove the default code of index.jsp

(b) extend Weatherclient ⇒ extend Web Service reference ⇒ extend Weather ⇒ extend Weather ⇒ extend WeatherSoap ⇒ drag GetCityWeatherByZip to index.jsp

```
//index.jsp
<%
  try
  {
    pack1.Weather Service1 = new pack1.Service();
    Pack1.WeatherSoap port = Service.getWeatherSoap();
    java.lang.String ZIP = "10007";
    Pack1.WeatherReturn result = port.getCityWeather
                                 ByZip(ZIP);
    String city = result.getCity();
    String temp = result.getTemperature();
    out.println("City = " + city);
    out.println("<br>");
    out.println("Temperature = " + temp);
```

3 catch (Exception e)

{

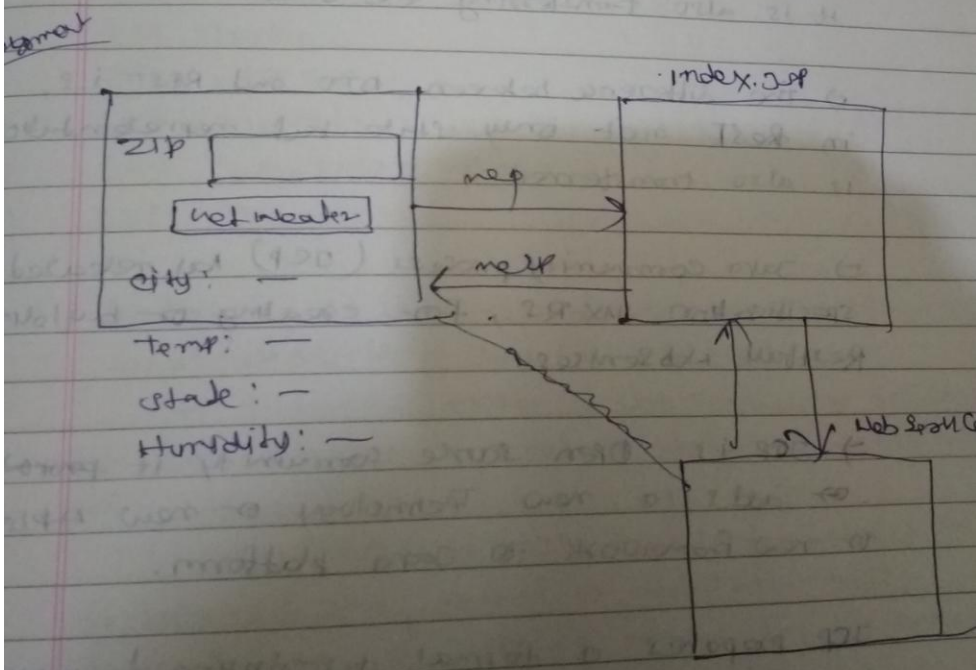
}

5) Right click on projectName → Run.

O/P

city = New York

temperature = 64



Date
01/05/19

Restful Web Services

RESTful

REST → Representational State Transfer.

Representation is nothing but a data format.
for eg.
text/html, application/json, text/xml etc

⇒ Representational state transfer (REST) is nothing but data transfer with representation.

⇒ in Java, we have data transfer object (DTO). it is also transferring the state.

⇒ The difference between DTO and REST is, in REST not only state but representation is also transferred

⇒ Java community process (JCP) has released a specification JAX-RS, for creating or building Restful Web Services.

⇒ JCP is, Open source community it adds a new Technology or new APIs or new framework to Java platform.

JCP prepares a formal description of proposed technology and after getting the comments on proposed document and after voting is done in JCP executive committee then final specification will be released.

For each final specification, a number is allocated with name Java Specification Request (JSR).

JAX-RS specification is also called JSR 311.

For JAX-RS specification, the implementation are provided by different vendors. Each vendor given implementation is called Restful container.

The following or list of Restfull container given below

(I.) Jersy
(II) REST Easy
(III) Restlet
(IV) Websphere Server
(V) Trioxis

Jersey is a Sun Microsystem given reference specification for JAX-RS specification.

In order to make a Web Service as accessible to the access internet, a Web Service must run on a Server.

in a Server, to create an object and to invoked the method of WebService class, a container support is needed in the Server.

⇒ Creating a RESTful Web Service

1) JAX-RS API has provided a set of annotations to create RESTful WebService.

(2.) JAX-RS has two packages
   a) Javax.ws.rs
   b) Javax.ws.rs.core

(3.) JAX-RS has two versions.
   a.) JAX-RS-1.x
   (b) JAX-RS-2.x

(4) In RESTful Web Services, each WebService class is called a root resource class and each method of WebService class is called a resource.

(5) The root resource class and resource file are identified by with relative Uri's.

(6) The Annotation used for adding a relative Uri to a root resource class and for a resource is @path.

eg.

@Path("/demo") → root resource / root relative path
(Optional)
public class Demo

{ @GET
  @Path("/hello") // sub relative path
  public String SayHello() - resource.

{
return "Hello---";
}
}

The RESTfull webservice classes and resources of the classes will be control or managed by a servlet given by RESTfull webservice container.

The servlet will be ~~vary~~ vary from one implementation to another implementation

The uri for accessing sayHello resource for the above ~~rest~~ RESTfull webservice is

http://localhost:4455/RESTApp1/resources/demo/hello

- RESTApp1 → project Name
- resources → url pattern of servlet
- demo → path to root resource
- hello → resource

eg2

```
@path("/demo")
public class Demo
{
    @GET
    @path("/hello/{uname}")          // uname → path variable
    public String sayHello(@pathParam("uname")   // → injecting a path variable to method parameter
                           String str)           // → ~~[illegible]~~
    {
        return "Hello" + str;
    }
}
```

The URI to access sayHello of for the above RESTful Web Service

http://localhost:4455/RestAPI/resources/demo/hello/sathya

O/P Hello : sathya

Eg:

```
@Path("/demo")
public class Demo
{
  @GET
  @Produces("text/html")
  @Path("/Welcome/{str1}-{str2}")
public String sayWelcome(@PathParam("str1")
                String str1, @PathParam("str2")
                String str2)
{
  return "<h1> Welcom : "+str1+" "+
          str2+" <h2>
}
}
```

URI

http://localhost:4455/RESTAPI/resources/demo/Welcome/sathya-sekhar

↓

Param, we use store on @Path above.

O/P Welcome : sathya sekhar.

egs

```
@path("/demo")
public class Demo
{  @GET
   @path("/{uname}")
public String sayWelcome(@PathParam("uname")
                         String str)
  {    return "Welcome : "+str;
  }
@GET
@ path("/hello")
public String sayHello(~~@PathParam~~)
{
        return "hello";
}
```

→ URI for calling the sayWelcome and sayHello web service is same

http://localhost:4455/REST App/ restws/ ~~hello~~ hello demo ↓

Here hello is suitable for uname and suitable for submitting path of hello method.

If the above uri a conflict is occur because it is suitable for sayHello and sayWelcome,

In order to avoid conflict, we use @QueryParam annotation.

@QueryParam annotation inject the value of query parameter to a method parameter.

eg

```
@Path("/demo")
public class Demo
{
    @GET
    @Path("/{more}")
    public String sayWelcome(@QueryParam("more")
                             String str)
    {
        return "Welcome :"+str;
    }

    @GET
    @Path("/hello")
    public String sayHello()
    {
        return "hello";
    }
}
```

Uri :

http://localhost:4455/RESTDemo1/webapi/demo?more=hello

@Path is optional for single method. If there are more than one methods then we have to add @Path

Here, we call sayWelcome

O/P

Welcome : Hello

V2

http://localhost/RESTApi/resources/demo/hello

O/P - Hello

Creating a RESTful WebService with eclipse and Tomcat

Tomcat server doesn't support any RestFul WebServices container. So, TO run a RESTful WebService on Tomcat we need to pass a restful WebServices container jar file to the server along with our application lib folder.

TO add Sun Microsystem given implementation of JAX-RS API (Jersey API)

- We need to download the jar file of Jersey container.

(1) Visit Jersey.Java.Net website and select download option

(2) go to Jersey 1.x section and click on Jersey zip bundle.

(3) extract the zip file, so that we will get a folder Jersey-archive-1.18

(i) create a dynamic web project in eclipse with name RestfulApp1

(ii) add JSR.311-Api.jar to the build path (Jersey-archive-1.19)

(iii) create a RestfulWebService class with the name DemoService like the following

```
// DemoService.java
@Path("/demos")
public class DemoService
{
    @GET
    @Path("/Welcome")
    @Produces("text/html")
    public String sayWelcome()
    {
        return "<h1>Welcome to RestfulWebServices</h1>";
    }
}
```

// ~~web.xml~~

configure a predefine Servlet given by the Jersey container in web.xml like the following

```
<!--web.xml
```

```
<!-- web.xml -->
<web app>
<servlet>
    <servlet-name> rest </servlet-name>
    <servlet-class> com.sun.jersey.spi.container.
                    servlet.ServletContainer </servlet-class>
</servlet>
<servlet-mapping>
    <servlet-name>
    <init-param>
        <param-name> com.sun.jersey.config.property.
                     packages </param-name>
        <param-value> com.satya.rest </param-value>
    </init-param>
<load-on-startup> 1 </load-on-startup>
</servlet>
<servlet-Mapping>
    <servlet-name> rest </servlet-name>
    <url pattern> /resources/* </url-pattern>
</servlet-Mapping>
</web.xml>
```

add the following JAR so the lib folder from Jersey zip file.

(i) asm
(ii) Jersey-client
(iii) Jersey-core
(iv) Jersey-server
(v) Jersey-servlet
(vi) ~~Jersey-api~~ JSR 311-API

Right click on project name ⇒ run as ⇒ Run on server.

Type the following request on browser

http://localhost:2014/RestfulWS/webapi/demo/welcome

3|14 Creating a RESTful Web Service with JAX-RS 2.X API

1) In JAX-RS 1.X API, We need to configured a Servlet provided by a JAX-RS container in web.xml file of the project.

2) Now, in JAX-RS 2.X, New annotation is introduced called @ApplicationPath. Because of this annotation we no need to configure a JAX-RS container provided servlet in a web.xml file.

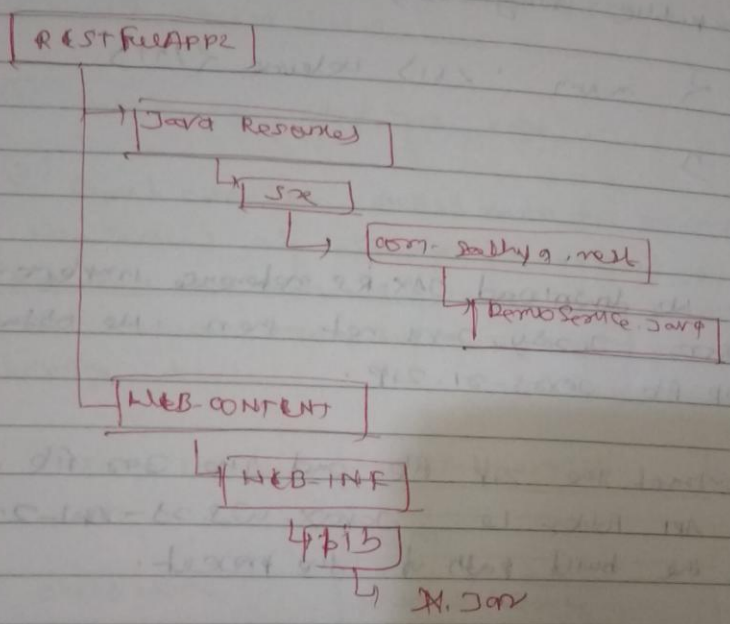
3) To the Application lib folder we need to attached the jar file which is provided by vendor.

Ex

JBOSS - implemented JAX-RS specification Java API to create RestFull Web Services and also support Rest WS container by default so we need not to add REST WS container

If we want to use Sun provided implementation of JAX-RS then we need to download JAX-RS 2.X jar file from Jersey.java.net website.

(4) The following example is a RESTFull application with JAX.RS 2.X




```
package com.sathya.rest;
import javax.ws.rs.ApplicationPath;
import javax.ws.rs.GET;
import javax.ws.rs.path;
import javax.ws.rs.produces;
import javax.ws.rs.core.Application;

@ApplicationPath("/rest")
@path("/demos")
public class DemoService extends Application
```

```
@GET
@Path("/welcome")
@Produces("text/html")
public String SayWelcome()
{
    return "<h1> Welcome </h1>";
}
}
```

- If we download JAX-RS reference implementation from Jersey.java.net then we obtain a zip file jaxrs-ri.zip.

- extract the zip file and the jar file available in API Folder i.e javax.ws.rs-api-2.0.jar to the build path of the project.

- Add the jar file from API and ext and lib folder to the our restful web app/lib folder.

- Deployee the application in Tomcate by selecting Tomcat server in run as application and from the browser type the following request.

http://localhost:8084/restful-ws-demo/webapi/welcome

creating a RESTful application in NetBeans IDE 7.3

File menu → New Project → Select Java Web → Web Application → Enter project name RESTful → next → Finish.

② Delete the index.jsp page.

③ Right click on project name → new → Others → Web Services at left side → RESTful Web Services from patterns at Right side → next → Select Style root resource radio button → next → Enter the following details

Resource package	com.sathya.rest
path	phone
class name	phoneBookService
MIME Type	text/html
Representation class	java.lang.String

click on Finish.

// phoneBookService.java

@Path("phone")
public class phoneBookService {
    @Context
    public phoneBookService() {

2

```
@GET
@produces("text/html")
@path("{uword}")
public string getphoneNumber(@pathparam("uword")
                        string username)
```

2

```
if (uword.equals("U1234"))
     return "1234567890";
elseif (uword.equals("U4321"))
     return "0123456789";
else
     return "sorry! uword is not valid";
```

3

2

right click on project & run

http://localhost:8085/RestDir/webresources/phone

U4321

↓

result

url pattern & resources path

Calling above phoneBook Servlet from console application at Java

Step

1) File menu ⇒ New project ⇒ Java ⇒ Java application ⇒ Enter project name (phoneClient) ⇒ uncheked create main class ⇒ finish.

(2) Right click on phoneClient project ⇒ New ⇒ Java class ⇒ Enter class name ⇒ (client) ⇒ finish.

```
// Client.java
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader
import java.net.HttpURLConnection;
import java.net.MalformedURLException;
import java.net.URL;
public class client
{
    public static void main(String args[])
    {
    try
    {
    URL url = new URL("http://localhost:
    4455/RestHTTP/WebService1/phone
    /9431");

HttpURLConnection conn = (HttpURLConnection) url.openConnection();
conn.setRequestMethod("GET");
```

```
conn.setRequestProperty("Accept", "text/html");
if (conn.getResponseCode() != 200) {
    throw new RuntimeException("Failed" + conn.
                                getResponseCode());
}
BufferReader br = new BufferReader(new
    InputStreamReader((conn.getInputStream())));
String output;
sop("output from server \n");
while ((output = br.readLine()) != null) {
    sop(output);
}
conn.disconnect();
// close the connection
} catch (MalformedURLException e) {
    e.printStackTrace();
} catch (IOException e) {
    e.printStackTrace();
}
}
}
```

convert it into BufferReader

convert it to InputStreamReader form

it return an object of getInputStream

⇒ Create client application using .net that for Java web, Properties service created in Java

Step

1) Start visual studio IDE ⇒ File menu ⇒ New ⇒ project ⇒ select v.c# ⇒ select console application ⇒ OK

(2) Defined a console application like the following

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Web;
using System.IO;
using System.IO;
using System.Net;

namespace ConsoleApplication1
{
    class program
    {
        static void Main(string[] args)
        {
            try
            {
                HttpWebRequest request = (HttpWebRequest)
                    WebRequest.Create("http://localhost:
                    4455/RestApp/webresources/hello/world");
HttpWebResponse response = (HttpWebResponse)request.
                    GetResponse();
```

```
StreamReader reader = new StreamReader(
                    .GetResponseReader(
                    GetResponseStream());

String message = reader.ReadToEnd();
Console.WriteLine("Response : " + message);
Console.ReadLine();

}
catch (Exception e)
{   Console.WriteLine(e);
    Console.ReadLine();
}
}
}
```

Working with RESTeasy container and JBOSS

(1) RESTeasy is a Restful container given by JBOSS Server

(11) RESTeasy container is build from JBOSS Server

(111) If we are working with other Servers other than JBOSS then we have to download RESTeasy container Jar file.

Json data represented by { } braces

(iv) Go to com download Resteasy s/w from the following website.

http://sourceforge.net/projects/resteasy/files/

Click Resteasy-jaxrs link

Click on 2.3.7 Final

click on resteasy-jaxrs-2.3.7.Final-all.zip

extract the zip file.

⇒ The following are steps to work with Resteasy Webapp and Tomcat using Eclipse.

step

(1) create a dynamic web project called RestApp1

(2) create a Restful Service class HelloService

```
package com.sathya.rest;
import javax.ws.rs.GET;
import javax.ws.rs.Path;
import javax.ws.rs.Produces;

@Path("/Hello")
public class HelloService
{
    @GET
    @Produces("text/html")
    @Path("/welcome")
    public String sayWelcome()
    {
        return "Hi welcome Hi";
    }
}
```

Add JAX-RS-API-version-3.0 from lib folder + resteasy jar to buildpath

define web.xml under WEB-INF folder like the following.

```
<web-app>
    <context-param>
        <param-name> resteasy.scan </param-name>
        <param-value> true </param-value>
    </context-param>
<! THIS need same with resteasy servlet url-pattern -->
    <context-param>
        <param-name> resteasy.servlet.mapping.prefix </param-name>
        <param-value> /rest </param-value>
    </context-param>
<listener>
    <listener-class>
        org.jboss.resteasy.plugins.server.servlet.
                    ResteasyBootstrap
    </listener-class>
</listener>
<servlet>
    <servlet-name> resteasy-servlet </servlet-name>
```

<servlet-class>
org.jboss.resteasy.plugins.server.servlet.
HttpServletDispatcher
</servlet-class>
</servlet>

<servlet-mapping>
<servlet-name>resteasy.servlet</servlet-name>
<url-pattern>/rest/*</url-pattern>

Copy all Resteasy jar file except resteasy-cdi jar file from lib folder of resteasy s/w to our web application lib folder

Right click on project name ⇒ Run as ⇒ Run on server ⇒ select ⇒ Tomcat ⇒ next ⇒ Finish.

Request

http://localhost:2014/RestfulAbhi/rest/Hello/welcome

Date
26/05/14

Exception Handling in RESTful Services

⇒ By Default, If an Exception is occured in a resource of a Restful Web Services then that Exception stackTrace message will be shown on browser and on server console.

⇒ Displaying Exception stackTrace message directly on browser is not a good idea because an end user is unknown about Exception of Java

⇒ To solve this above problem, we need to send a Meaningful Message on browser, instead of displaying Exception stackTrace Message.

⇒ In RESTful Web Services, if we want to handle an Exception thrown by a resource (method) of a Web-Service class (not resource class) then we need to define an Exception Mapper class.

⇒ In RESTful Web Services, an Exception Mapper class can be created by implementing our class from Exception Mapper interface.

eg

public class MyExceptionMapper implements ExceptionMapper<MyException>

{

@Override

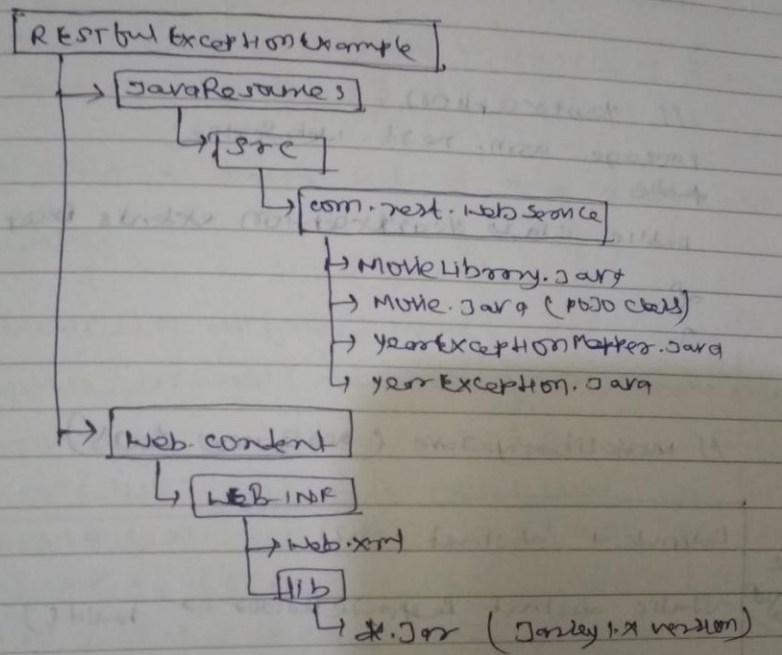
public Response toResponse(MyException)

{

}

The following RESTful Web Service Application is to return a list of movies released in given year.

If The given year is not in a range instead of exception a message will be send to the client, by handling the exception



// Movie.Java

```
package com.rest.WebService

public class Movie
{
    private int year;
    private String title;

    public Movie() {}
    public Movie(int year, String title)
    {
```

this.year = year;

this.title = title;

}

// setter and getter

public int getYear() {}

}

// YearException

package com.rest.webservice;

public

public class YearException extends Exception

{

}

// MovieLibrary.java (webservice class):

Response : abstract class

static abstract ResponseBuilder → build()

enum Status

ResponseBuilder is nested class of Response

Status is enumeration which takes parameter as an object of Response.

```
// MovieLibrary.java
package com.rest.webservices

@Path("/movie")
public class MovieLibrary
{
// Store the in memory list of our repository.
static final List<Movie> MOVIE_LIST = new
ArrayList<Movie>();

// Build a dummy list of movies to work with.
static
{
MOVIE_LIST.add(new Movie(2008, "Gandhi"));
MOVIE_LIST.add(new Movie(2008, "Jodha Akbar"));
MOVIE_LIST.add(new Movie(2013, "Dabangg"));
MOVIE_LIST.add(new Movie(2011, "RA.One"));
MOVIE_LIST.add(new Movie(2011, "RODO"));
}
@GET
@Produces("application/json")
@Path("/{year}")
public List<Movie> getMovies(@PathParam("year") int year)
throws Exception
{
if(year < 2008 || year > 2013)
{
// Invalid input for year so throw exception
throw new YearException();
}
}
```

```
List<Movie> list = getMoviesByYear(int targetYear);
{
    List<Movie> found = new ArrayList<Movie>();
    for( Movie movie : MOVIE_LIST)
    {
    if( movie.getYear() == targetYear)
        found.add(movie);
    }
    return found;
  }
}
```

// YearExceptionMapper.java

```
package com.rest.webservice;
@Provider // it is helper class to a WebService
          class, it is annotated with (@Provider)
public class YearExceptionMapper implements ExceptionMapper<YearException>
{ public Response toResponse(YearException arg0){
    List list = new ArrayList();
    list.add("Year is not valid");
    list.add("year should be >=2000 and <=2015");
Response.Status httpStatus = Response.Status.BAD_REQUEST
```

↓
set the final
variable to another

```
ResponseBuilder builder = Response.status(
        HttpStatus)

builder.entity(list);
Response response = builder.build();
return response;

}

}

// web.xml

<web-app>
  <servlet>
    <servlet-name>rest</servlet-name>
    <servlet-class>com.sun.jersey.spi.container.
        servlet.ServletContainer</servlet-class>
    <init-param>
      <param-name>com.sun.jersey.config.
          property.packages</param-name>
      <param-value>com.rest.WebService</param-value>
    </init-param>
    <init-param>
      <param-name>com.sun.jersey.api.json.POJOMapping
          Feature</param-name>
      <param-value>true</param-value>
    </load-on-startup>1</load-on-startup>
  </servlet>
```

```
<servlet-mapping>
<servlet-name> rest </servlet-name>
<url-pattern> /rest/* </url-pattern>
</servlet-mapping>
</web-app>
```

* add jersey311.all.jar to build path of project
* copy jersey1.x.jar file to the [lib] folder of the project

⇒ Request:

localhost:2014/RestExercise0Example/rest/movie/2008

o/p

{ [{"year": 2008, "title": "Bandhi"},

{"year": 2008, "title": "Jatra"}]

Request2

localhost:2014/RestExercise0Example/rest/movie/abc

o/p

["year is not valid", "year should be >= 2008 and <= 2013"]

Request3

localhost:2014/RestExercise0Example/rest/movie/2009

o/p []

Executing a RESTful Web Service on JBoss 6.x

(1) From JBOSS 6.x version, the server contain built in RESTeasy RESTful WebService container and it is implementation of JAX RS 2.x specification.

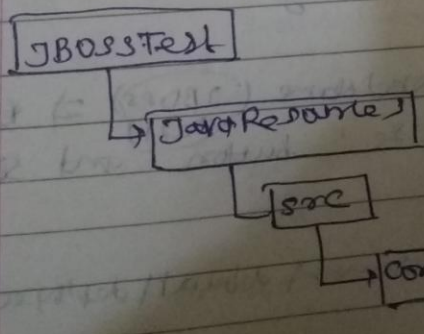
(2.) We no need to include the resteasy Jar files and web.xml configuration in a restful webservice application.

(3.) Download JBOSS-6.1 Server zip file and extract it so that server is installed in the machine.

(4.) By default, JBOSS HTTPPort number is 8080 and it will clash with oracle HttpServer. so change http port number like the following.

C:\Jboss-6.1.0.Final\Server\default\conf\bindingservice.beans\META-INF\bindings-jboss-beans.xml and change port property of ServiceBindingMetaData bean in the xml.

(5.) Create a RESTful WebService in Eclipse like the following



```
package com.sathya.rest;
import javax.ws.rs.ApplicationPath;
@ApplicationPath("resource")
@Path("sample")
public class SampleService extends Application
{
  @GET
  @Path("hello")
  @produces("text/html")
  Public String sayHello()
  {
    return "<font color = red> Hello</font>";
  }
}
```

Eclipse ide (including latest Kepler) has not given adopter for adding JBOSS 6.x to the ide. So we need to export our application as War file into JBOSS like the following.

(1) Right click on projectName (JBOSS) ⇒ Export ⇒ War ⇒ click on 'browse' button and select the destination as

C:\JBOSS 6.x-final\server\default\deployee folder

⇒ Save ⇒ finish

(2) Go to JBoss bin folder and double click on batch file [run] then server will be started.

(3.) Type the following request URL from the browser

http://localhost:2015/JBossTest/resource/sample/hello

Securing a RESTful Web Service

1.) A RESTful Web Service can be access by any client with the help of its URI. If we want to make a RestFul Web Services as a protected resources then we need to apply security for the Web Service

(2.) To apply security we need to configure Security related tags in web.xml file.

(3) While developing a RESTful JAX-RS with 2.x version we don't required web.xml for a web service but to configured the security, we need to create web.xml with security tag

eg:

```
<!-- Security configuration -->
<security-constraint>
<web-resource-collection>
<url-pattern>/resource/demo/*</url-pattern>
</web-resource-collection>
<auth-constraint>
</security-constraint>
```

<role-name> admin </role-name>
</auth-constraint>
</security-constraint>
<login-config>
<auth-method> BASIC </auth-method>
<realm-name> my_realm </realm-name>
</login-config>

in the above security configuration BASIC authentication method is used it means dialog box is prompted to enter the user name password, while accessing protected resources of the server.

for eg, while accessing tomcat manager application, dialog box is prompted to enter a username and password. It is basic authentication model.

under a role, one and more user and password is exist.

TO configure, username and password of a role we need to open Tomcat-user.xml of conf folder.

<User name = "sekhar" password = "java"
roles = "admin" />

<User name = "java" password = "sathya"
roles = "faculty" />

1

Q) If more than user are accessing the RestFul Web Service then if one user change the resource of Web Service then it can be effect on other user or not.

A) No, Because RestFul Web Service is a stateless. The object is create for each request and it is light weight.

D/W b/w SOAP Based Web Service and RESTful Web Service.

What is SOAP fault ? What is three important tag of soap request and soap response.

If any exception is occured then exception stack print trace message is send to client in the form of <soap: fault> and this <soap: fault> contains <soap: Details> and <soap: message>

⇒ Exception Handling in a SOAP Web Services:-

1) While invoking an operation of a SOAP web service, if any exception is occured in operation at server side then that exception stackPrintTrace Message will be send as response of the SOAP response document back to client.

2) The SOAP response document insert under <SOAP: fault> tag of SOAP response body. It means an exception of a SOAP WebService will be mapped to or converted to <SOAP: fault> tag.

<SOAP: Envelop>

SOAP Response document

<SOAP: Envelope>

<SOAP: Header/>

<SOAP: Body>

<SOAP: fault>

<SOAP: details>

<SOAP: message>

=

</SOAP: message>

</stack details>

</SOAP: fault>

</SOAP: Body>

</SOAP: Envelope>

eg. WebService is in Jars and client is in .Net. If exception is occured in Java WebService then Java exception stackTrace will be send to the .Net client as a soap fault. but .Net client application cannot understand Java exception. so a problem occured.

In order to solve the above problem, the exception in a Soap Based Service must handled and response will be send as a meaningful message under <soap:fault> tag of soap response document

⇒ The following application is soap based WebService with Exceptional Handling.

(1) The WebService contains one operation called sayHello and it throws simple Exception if invalid Send by the client application as Hello.

(2) in a soap exception Handling, to handle the exception, we need to create a class by adding @WebFault annotation on TOP or by extending the class from Exception.

(3) We need to create a POJO class to store the exception message of an exception

Step:

(1) Start NetBeans

(2) File → New Project → Java Web → Web Application →

next → project name → soapExceptionTest) → Finish
→ double index) → right click on soapExceptionTest
→ New → Java class → class name (SimpleExceptionBean
→ package (com.sathy.soap) → Finish

SimpleExceptionBean.java

```
package com.sathya.soap;
public class SimpleExceptionBean
{
    private String message;
    public SimpleExceptionBean() {
    }
    public SimpleExceptionBean (String message)
    {
        this.message = message;
    }
    public String getMessage() {
        return message;
    }
}
```

3) Again click on soapExceptionTest → New → Java class → class name (SimpleException) → package (com.sathya.soap

```
package com.sathya.soap;

@WebFault
public
~~package~~ class SimpleException extends Exception
{
  private SimpleExceptionBean faultBean;
  public SimpleException(String message, SimpleException
                         Bean faultInfo)
  {
    super(message);
    faultBean = faultInfo;
  }
public SimpleExceptionBean(String message,
                SimpleExceptionBean faultInfo, Throwable
                cause)
  {
    super(message, cause)
      faultBean = faultInfo;
  }
public SimpleExceptionBean getFaultInfo() {
  return faultBean;
  }
}
```

Right click on projectName → Other → WebService
→ WebService → Enter (WebServiceName (SimpleWebService)
→ Package (com.sathya.soap) → Finish.

SimpleWebService.java

```
@WebService(serviceName = "SimpleWebService")
public class SimpleWebService {
    @WebMethod(operationName = "SayHello")
    public String SayHello(@WebParam(name =
        "name
{
    if(name.equalsIgnoreCase("Hello"))
    {
        throws new SimpleException("Story Hello is
            not correct", new SimpleExceptionBean());
    }
    return "Hello : "+name
    }
}
```

Right click on project name → Deploy
To see the WSDL file of the web service → Type the following request in address bar http://localhost:4455/SoapExceptionTest/SimpleWebService?wsdl

New project →

Right click on Java → Java application → main.java → SoapExceptionClient → main(String[] args)

Finally

right click on SourceCode client → new → other
→ Web Service → Web Service Client → enter
WSDL URL (http://localhost:4455/SourceCode/
/SimpleWebService?wsdl → package p1 → finish.

⇒ Right click on projectName (CC → New →
Java class ⇒ Enter class name (client) ⇒
Package (Optional) → Finish.

⇒ copy p1 package from generated source to
source package. ⇒

```
public class client
{
    public static void main(String args[])
    {
        try
        {
            SimpleWebService_Service service =
                new SimpleWebService_Service();
            SimpleWebService port = service.getSimpleWebServicePort();
            System.out.println(port.sayHello("Hello"));
        }catch(Exception e) { System.out.println(e); }
    }
}
```

→ Right click on program Space here click
on run file.