Rs = 100/-

WEBSERVICES NOTES

BY

SHEKHAR SIR

SATYATECHNOLOGIES

SRI RAGHAVENDRA XEROX

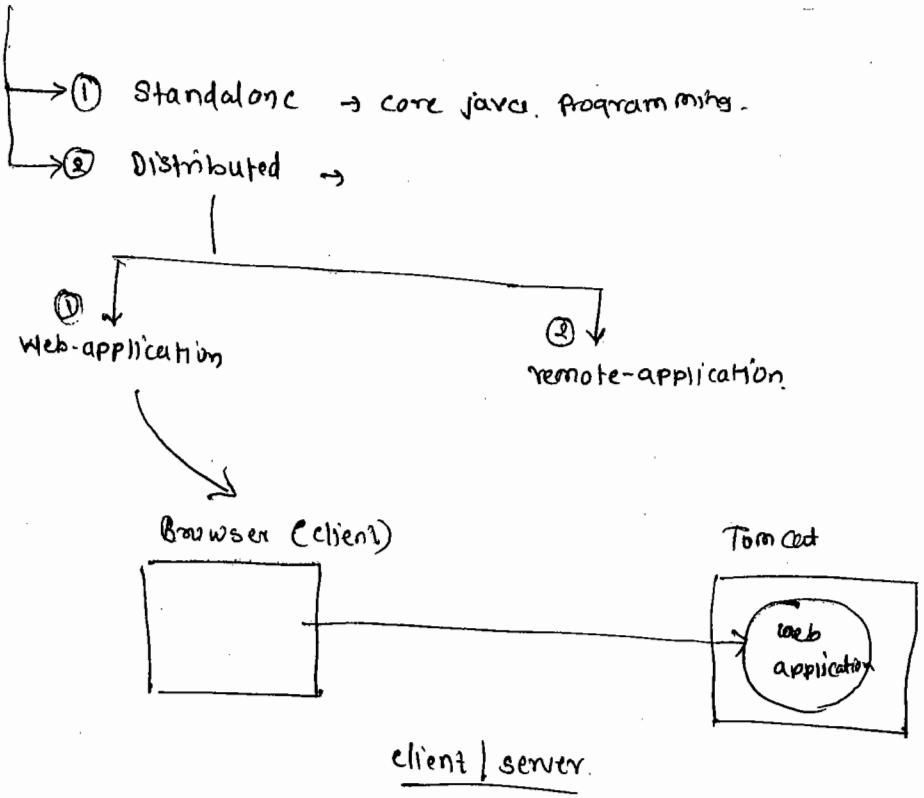
Software Languages Material Available

Beside Bangalore Ayyangar Bakery, Opp. C DAC, Ameerpet, Hyderabad.

Cell: 9951596199

16-4-2015

Java Applications



① Networking API (Socket Programming)

Problem → "location transparency"

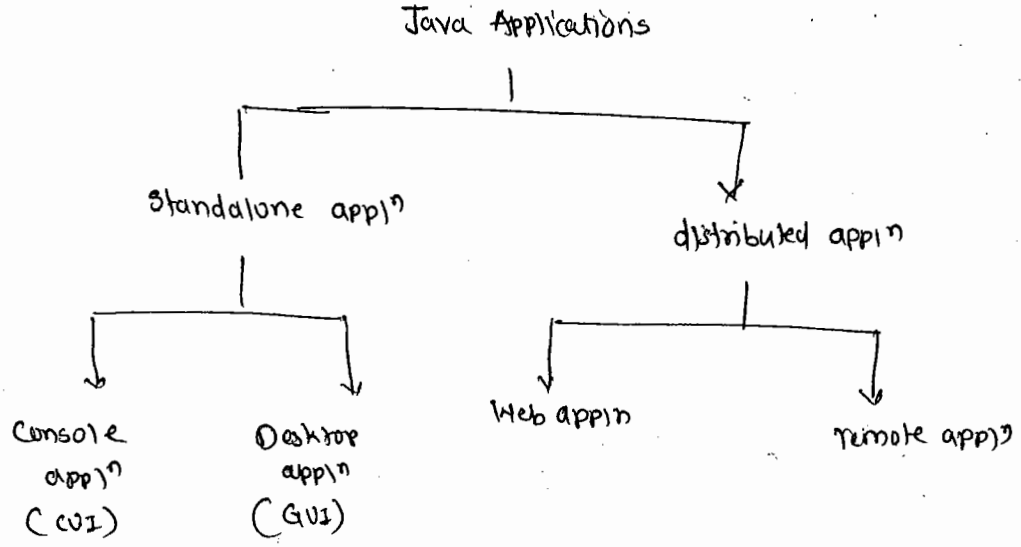
② RMI

③ CORBA

④ EJB

⑤ Webservices → for remote application.

Java WebServices



→ Standalone Applications are created using Java SE.

→ In real time standalone applications are ~~used~~ developed very rarely.

→ all real time appln's are distributed Appln, it means they follow client/server model.

→ a [Web-application] is also a distributed application, where application runs on a webserver and it is accessible through browser as a client.

→ a [remote application] is also a distributed application, where Server application is accessible through a client application, not a by using browser.

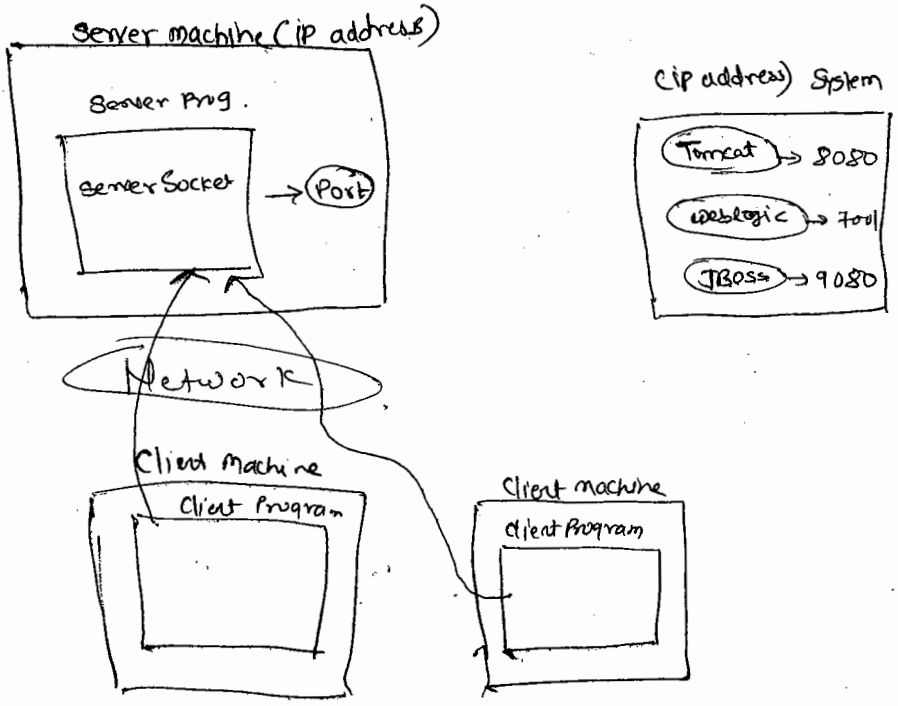
*** Q What is a difference between a web application and Remote Application.

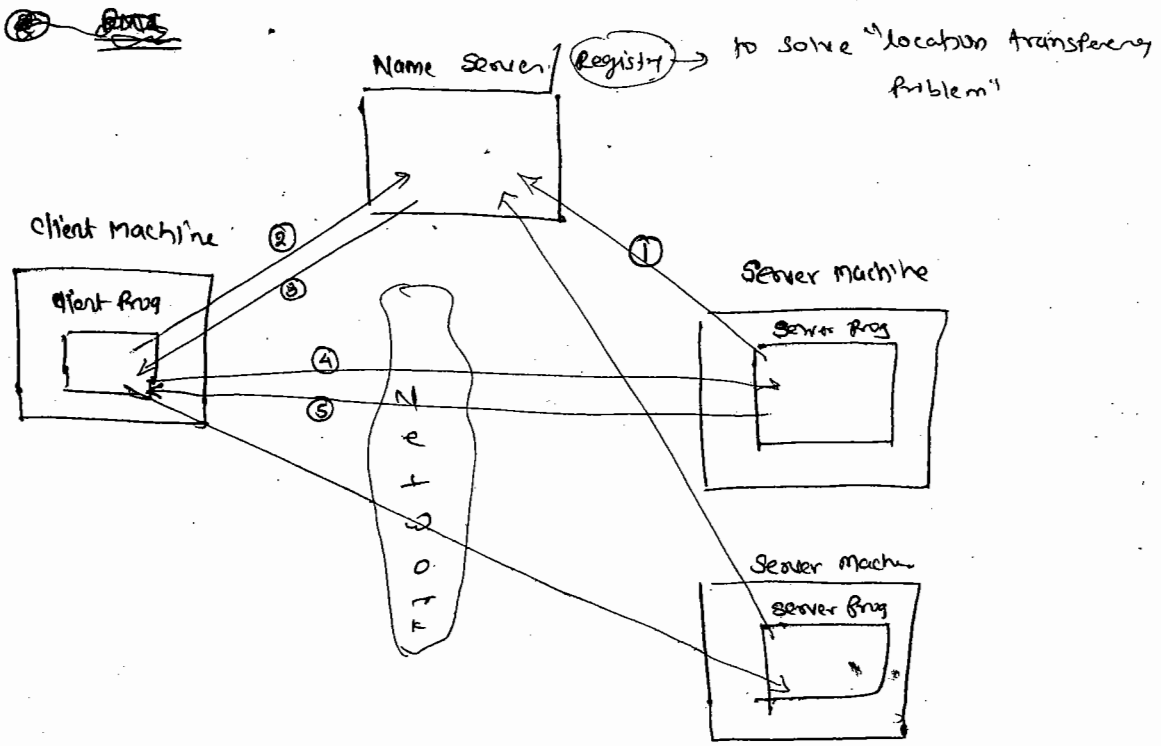
Web application	Remote Application
① In a web application client is a **browser.**	① In a remote application client is a **Java Application.**
② In web applications server provides a service directly to the end user	② In a remote application server provides service to client application & then client Application will provide service to end user.
③ In a web application no need to install Java software at client side a browser is enough.	③ In remote application, at a client side we need to install Java software, to run the client application.

19-4-2015

Problems with:

1) Socket Programming → "location transparency problem"





→ **different ways of Developing Remote Applications in Java**

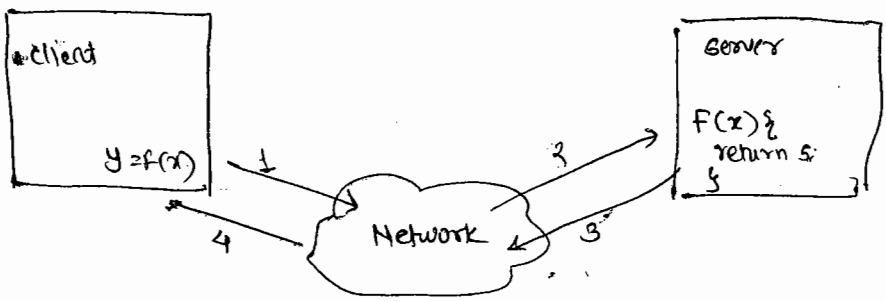
① Socket Programming.

→ Socket programming is also called Networking API.

→ Socket programming is a Part of Java SE.

→ In socket programming client application connects with a server application using IP address of server machine and Port number of a server application

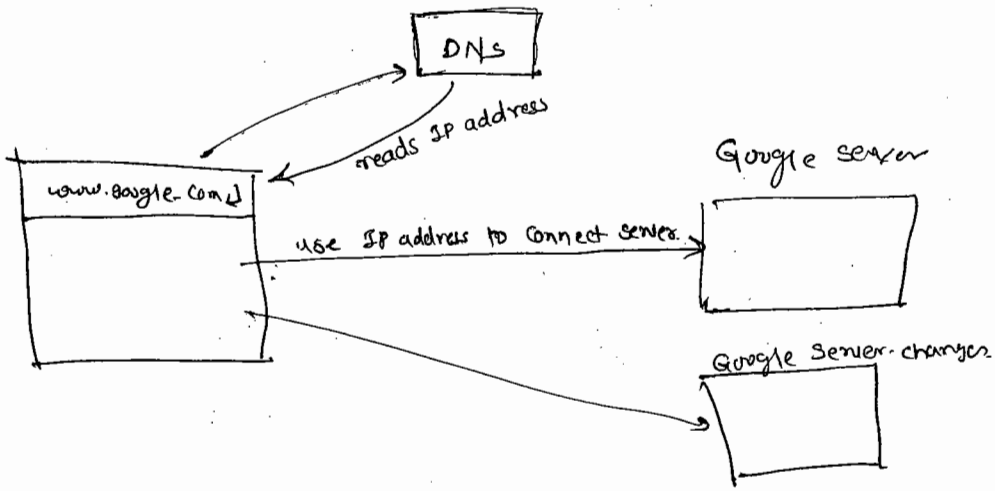
Client Server interactions:



1. Send message to call f with parameter x
2. Receive message that f was called with the given parameter.
3. Send message with the result of calling f.
4. Receive message with the result of calling f.

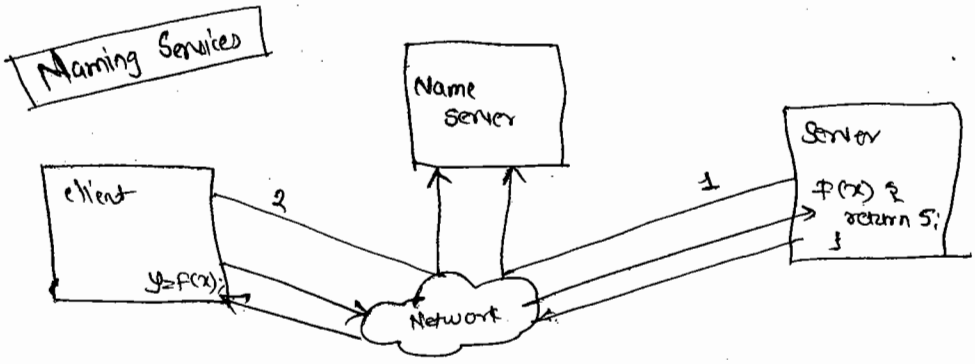
→ In socket programming a problem is identified i.e. "Location transperency"

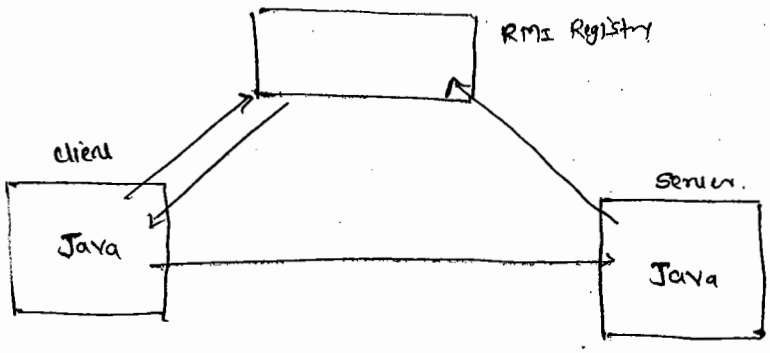
→ "Location transperency" Problem means if the location of the server application is changed from one system to another then client application can not connect with server application



→ to resolve the location transperency problem, in the middle of client and the server a name server is introduced

→ a client application will get the location of the server from Name Server before connecting to server application.





— small API
- Non
→ RMI is distributed Homogenous Protocol. technology

RMI Protocol (Not firewall friendly)

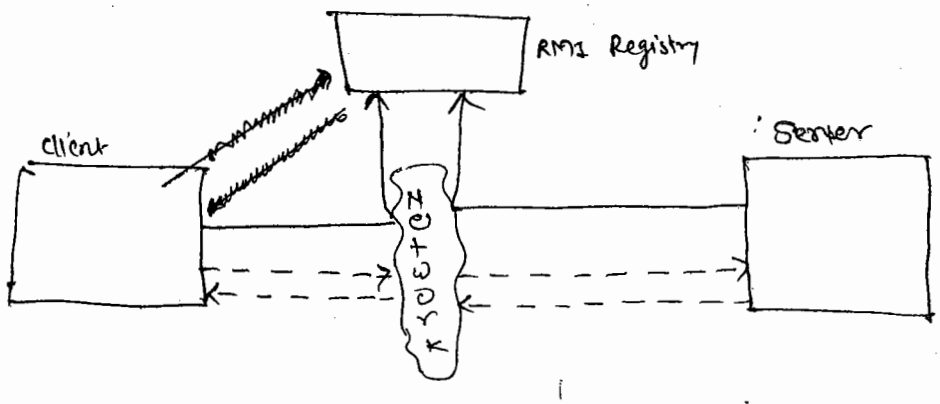
② RMI (Remote Method Invocation):

→ RMI API is released by Sun as part of Java SE

→ RMI API is given to developed client and server Applications in Java.

→ with RMI, RMI Registry is introduced to act as mediator between client and server applications.

→ In RMI comm^n a server application registers with RMI Registry and the client application connects with RMI Registry to find the Server and then client appl^n connects with server appl^n



→ **following Problem are Identified with RMI.**

① → RMI is the small API & it can be not be used for developing large Applications.

② client & server Appl^n in RMI connects with RMI Registry with RMI Protocol, but it is not a firewall friendly Protocol.

③ RMI is only for Communicating Java to Java. it means RMI is a distributed Homogenous Technology

20-4-2015

④ CORBA

⑤ **CORBA** Common Object Request Broker Architecture

→ corba is a technology from object management group. (OMG).

→ CORBA is a distributed Heterogenous technology for developing Remote Appl^n

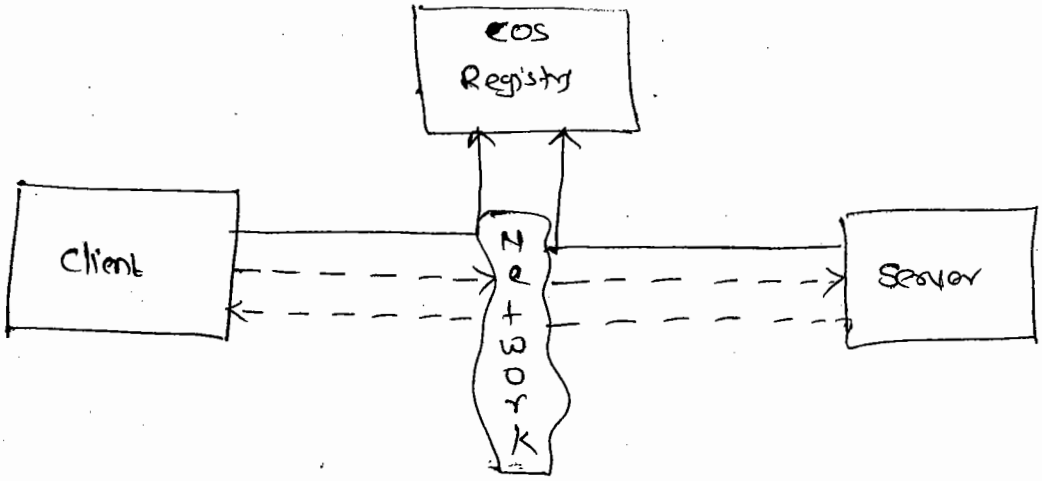
→ CORBA Provides comm^n bet^n two appl^n developed in different Programming languages, running on diverse platforms.

→ to Provide comm^n bet^n two different languages application. CORBA introduce a neutral language called IDL (Interface definition language.

→ a server application should create its interface in IDL. TO Create interface in IDL, CORBA has provided mappings bet^n IDL & other languages.

→ At client side, to convert IDL interface into a client side languages, CORBA has provided Compilers.

→ A client and server uses COS Registry as a mediator for communication.



→ CORBA is not accepted by the industry, because IDL syntaxes are complex & IDL has not given all the Datatypes to map with particular language.

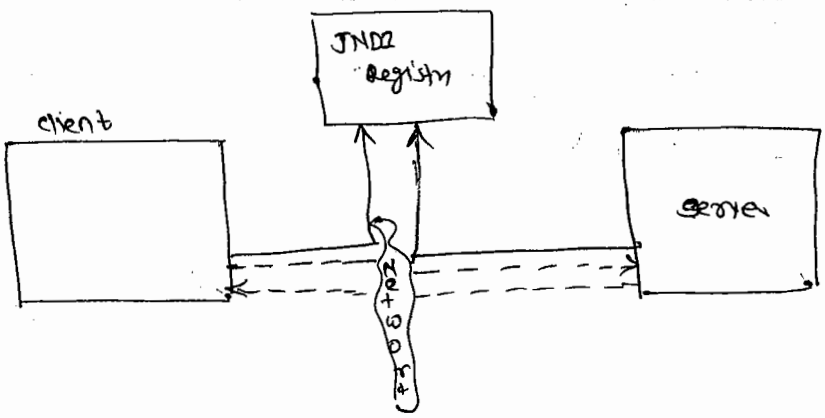
EJB
↳ large API
→ need to installed Glassfish, Weblogic server for EJB container
→ Homogenous technology

(4) EJB

→ EJB is a open source technology from sun microsystem
EJB API is very large API for developing Remote application

→ In EJB bet^n client & server application JNDI Registry (Java naming Directory & Interfaces) is a mediator

→ EJB is a Homogenous technology it means a Comm^n bet^n the Java client appl^n to Java server Appl^n.



→ the problems with EJB.

① it is a Heavy weight technology

② It is a distributed Homogenous technology.

⑤ Web Services

→ In order to develop interoproble client server applications, an organisation with professionals of microsoft, sun, IBM, Oracle etc. has formed liked one group called (WS-I) web services Interpreble.

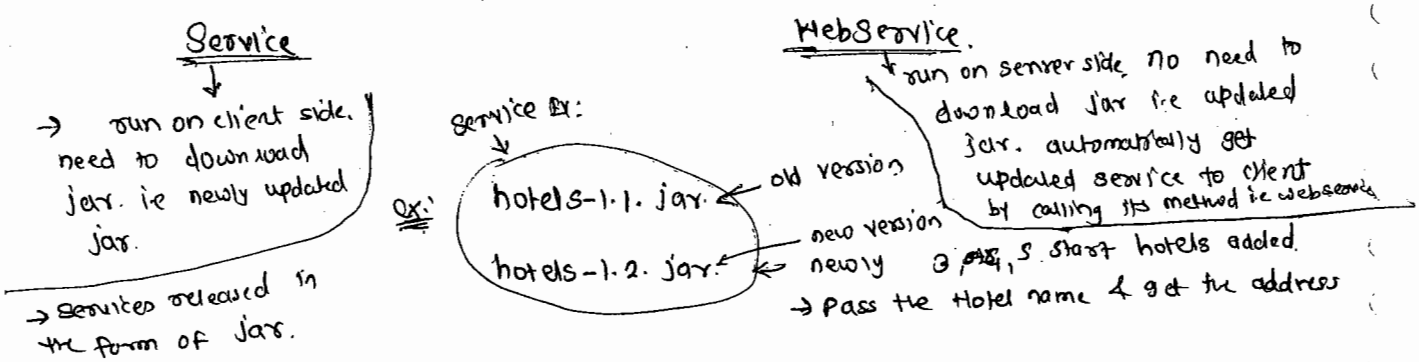
→ webservices is a technology ~~called~~ for developing distributed Helenogenous technology.

→ webservices is the 2nd distributed Heterogeneous technology after CORBA.

→ with web services technology a client and server application developed in different programming lagauges. Posted on different servers, created using different technology and

and running on different Platform can communicate with each other this called Interoperability.

21-4-2015



Difference betn Service & webservice

→ A service is a Program, which Provides services to other Programs

→ We can take a service like a class and it Provides to functionalities to other class

→ In Java, a developer or a vendor can Provide the services to rest of the developers. by releasing the api in the form of a Jar file.

→ for example, iText is a vendor Provides Pdf service to Web application developers, released iText api in the form of iText.jar

→ A Problem with releasing services as Jar files is, it

any new features are added to the service and released another version of jar file then a client-side application developers should download new jar file.

→ In case of webservices, a webservice will also provide services to the client-side applications, but it will not be released in the form of a jar files.

→ a webservice will run on its server side only, a client across Network can access webservice.

→ ~~and~~ an advantage of webservice approach is, if any new features are added to a webservice then a client can access them directly, by without downloading any jar files.

Q What is a difference between a web Appl^n & web service.?

→ the intention of Web application always is to provide service directly to the end user.

→ the intention of web services is to provide services to the client Applications, but not to the end-user directly.

Basic Terminology

Basic Profile.

WS-I : BP-1.0 specification

WS-I → organization

BP-1.0 → documentation

specification → based on specification SUN releases JAX-RPC API

using this api we can develope webservice.

BP - 1.1 Specification

↓

SUN → (JAX-WS API) new

Used to create webservice & to create client ~~for API~~ for calling webservice ~~for Interface~~

24-4-2015

Basic terminology used in Webservices

WSDL → webservices discription language
→ is a xml file
→ operation, i/p, o/p, location (url of the service provider)

RMI/EJB → Homogenous.

client (Java) | interface (Java) | Service. (Java)

CORBA

client (any lang.) | interface (IDL) | service (any lang.)

Webservice :

client | Interface (WSDL) | Service (any)

WSDL

↓

What He tells

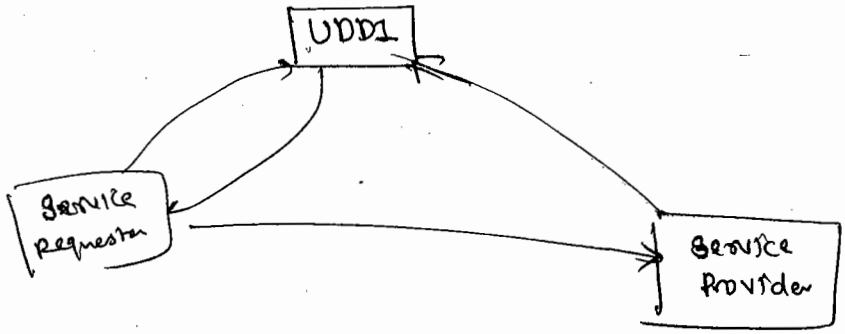
① Operation
② i/p
③ o/p
④ Location of Webservice

xml → independent to all language

.WSDL file put in Registry (UDDI)

Client need to download this

~~→ Url for~~ or url to download



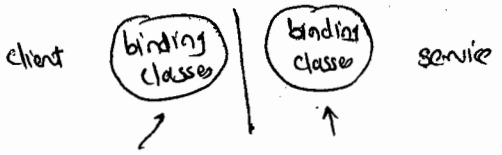
④ SOAP :- → Simple object Access Protocol.

→ is kind of xml.

SOAP request → xml (input)

SOAP response → xml (output)

— because of SOAP xml no dtd



→

① Basic Profile

→ WS-I organisation has released a specification document with a set of rules and guidelines to create APIs by the language Providers.

→ A language provider should prepare API for creating a Webservice or a client for calling a webservice in that language.

→ the specification released by WS-I organisation is Basic Profile. Its old version 1.0 and current 1.1

→ SUN microsystem has released JAX-RPC [Java API for Xml - Remote Procedure call] API for BP-1.0 specification

and JAX-WS [Java API for xml-webservice] API for BP-1.1

→ By using JAX-RPC API & JAX-WS API, we can create a webservice or a client for webservice in Java language.

② WSDL (WebService Discription Language) :-

→ In a webservices service and its client applications can be in different languages so a service should Provide Information about its operations to the client applications. In a mediator language.

→ WSDL is a language of type xml given by WS-I, for Providing Information about a service to the clients

→ Mainly a wsdl file tells the following information about a service to the clients, in the form of xml.

① Methods (operations) of webservice.
② input for operations.
③ output for operations.
④ location of service in network.

③ UDDI (Universal Discription Discovery and Integration) :-

→ In remote application, a registry is required, before actual Communication takes Place.

→ In webservices, a service Provider will store a WSDL file of service in a registry called UDDI.

→ a client will download WSDL file from UDDI registry, before connecting with a service.

→ UDDI registry is not a very much used in a webservices. In place of it, a wsdl file location will be placed in a website.

④ SOAP (Simple object Access Protocol):

→ Web services is a heterogenous communication technology. so client and server application can be different languages.

→ while sending a request / receiving a response a Problem occurs, due to datatypes mismatching or memory size mismatching.

→ To solve the above Problem, WS-I organization has given another xml language called SOAP.

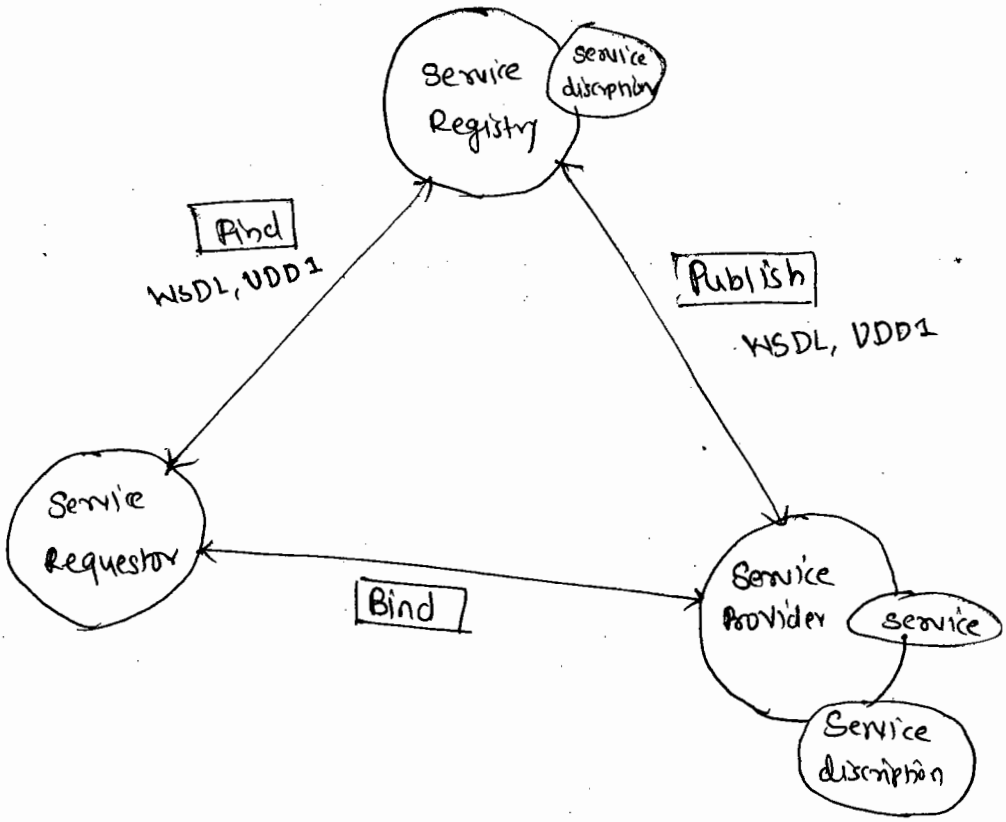
→ Between client and server, the request and response are transferred in the form of SOAP xml document. so there will be no Problem, because xml is a universal language for all other languages

25-4-2015

① Service Provider. → should create a service.
② Service Requester. } roles
③ Service registry

a) Publish → storing WSDL
b) find } operations
c) bind

Web Services Architecture (Service oriented Architecture (SOA))



Web Service architecture (SOA) has two components.

1) Roles
2) Operations

Roles

1) Service Provider:

→ Service Provider is an entity that defines a web Service.

→ a service provider will do the following:

a) creates a description for the service.

b) deploys the service in a runtime environment (web / application server) to make it accessible across network.

c) Publishes the service description to service registry.

② Service Registry

→ a service registry is to enable match-making between service Provider and service requestor.

→ Once match has found then interactions are carried out directly between a service requestor and a service Provider.

→ A Service Registry will do the following.

a) accepts request from service providers to publish & advertise web service descriptions.

b) allows service Requestors to search for service descriptions contained within the service Registry.

③ Service Requestor

→ A service Requestor is a component who consumes a webservice over a network.

→ A Service Requestor will do the following

a) finds a service description published in a service registry.

b) uses the service description to bind and invoke the webservice hosted by the Provider.

Operations

a) **Publish:**

→ the Publish operations performs service Registration or service advertisement.

→ when a service Provider Publishes its Web Service in a registry then it is advertising the service to all requestors in a network.

b) **Find**

the find operation performs searching for a service based on some condition. The result is a service descriptions that matches the search criteria.

c) **Bind:**

the Bind operation creates a client/server relationship between a service requestor and service provider.

XML (Extensible Markup Language)

GML → IBM
↓
SGLML → W3C
HTML XML

GML → Generalized markup language given by IBM
↓ donated to W3C
SGML → Standard Generalized markup lang.

XML → Meta markup language.

hbm file

<hibernate-mapping>

=

</hibernate-mapping>

XML technology
tags.

hibernate
↓
markup language
mapping

DTD/XSD

XML contains some tag, datatypes, keywords to create a another markup-language of type XML

web.xml (document) ← Vendor created ← XSD/DTD
↓
Programmer

Struts-config.xml

<struts-config>

</struts-config>

Apache
↓
created one markup lang. of type xml using tag given by xml technology.
(DTD)
tags

→ XML is a technology, used for transferring a data between two applications in a language independent & Platform independent manner.

→ XML and HTML both are derived from SGML.

→ for XML technology W3C organizations has released a specification with the setup of tags, keywords and datatypes for creating markup languages of type XML.

→ XML is a not a markup language, it is a meta markup language.

→ Vendors creates markup languages of type XML by using XML specification.

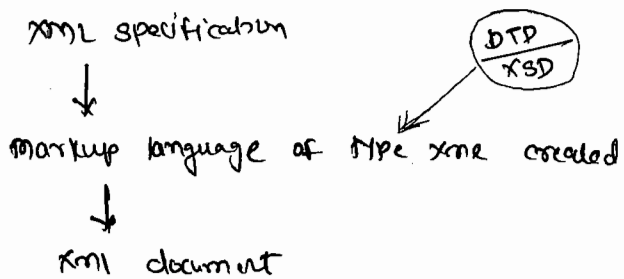
→ Vendors created markup language will be used by developers or programmers for exchanging or transferring data.

27-4-2014

XML

[Meta markup language]

using XML specification



Xml is neutral lang. for transferring data from one appl^n to other

→ XML Specification given by W3C is used for creating a markup language of type xml. and the that markup language will be used to creating xml document.

→ xml is mainly used for exchanging of data between a client and server applications.

→ a Server application should tell a client application about the tags, attributes, and the order of tags for exchanging the data.

→ In order to tell the clients about the structure of an xml document a server side vendor will prepared a markup language of type xml in the form of a DTD/XSD.

```
<web-app>
    <servlet>
        <servlet-class>xxxx</servlet-class>
    </servlet>
    <servlet-name>xxx</servlet-name>
    <url-pattern>/srvt</url-pattern>
</web-app>
```

Well-formed XML document

→ xml documents are 2 types

① Well-formed XML document.
② Valid XML document.

→ a XML document is said to be well-formed if it follows the below rules.

① a document should contain one and only one root tag or element.

② every open tag should has respective closed tag.

③ if a Parent and child tag are open then while closing child tag should be closed first & then Parent tag.

④ attributes values must be quoated (" ", ' ').

→ a xml document is valid document, if it follows the structure of xml defined by service provider in the form of DTD or XSD file

→ a well-formed xml document may not be a valid, but every valid xml document is also a well-formed.

two types of XML Parser contain in browser, but only Non-validating contains in browser not validating xml parser.

① Non-validating → well-formness

② validating → well-formness + validity.

Every browser contain xml browser. which is ~~non-~~ (not-validating Parser)
In Project we are using validating xml Parser.

Validating Parser.

SAX → simple API for xml.

DOM → Document object module.

Stax → Java 6

→ ~~XSD~~

XML Parser

→ an xml Parser is a program, it verifies Well-formedness and validness of XML document.

→ xml parser are 2 types
  ① non-validating parser.
  ② validating Parser

→ a non-validating parser will only checks for Well-formedness of xml document.

→ every web browser contains a non-validating Parser.

→ a validating parser checks for both well formness & validness of XML document.

→ In Java validating parser are 3 types.
  ① SAX → Simple API for xml.
  ② DOM → Document object model.
  ③ StAX → Streaming API for xml.

Document Type Definition (DTD)

Create xml file in the form of DTD/XSD

root
→ <addresses>
Parent → <address>
child
<hno>
<street>
<city>
</address>
</addresses>

addresses.dtd

```
<!ELEMENT addresses (address+)>
```

- `<!ELEMENT` — given by XML technology to create root tag.
- `addresses` — root tag.
- `(address+)` — Parent tag; `+` = one or more than one parent tag.

```
<!ELEMENT address (hno, street, city)>
<!ELEMENT hno (#PCDATA)>
<!ELEMENT street (#PCDATA)>
<!ELEMENT city (#PCDATA)>
```

`(hno, street, city)` — child tag

assigned PCDATA if there is not subelement for that element

Markup language created in the form of DTD

- `*` → element repeated 0/ more time.
- `+` → element repeated one or more than one.
- `?` → zero/one time

Occurrence operator or Cardinality operator.

client

a.xml

```
<addresses>
    <address>
        <hno> — </hno>
        <street> ammanpur </street>
        <city> Hyd </city>
    <address>
    <address>
        <hno> — </hno>
        <street> — </street>
        <city> — </city>
    </address>
</addresses>
```

Parser check the xml file.

Parsed character DATA

#PCDATA

- `#` — representing keyword in xml
- `PCDATA` — Reserved Keyword

character DATA

CDATA

XML technology → XML technology which contain some tag, datatypes, keyword to create a another markup-language of type XML.

XML is Meta-markup language.

Vendor created → Using this tag, datatypes, keyword Vendor created markup language in the form of DTD or XSD.

DTD is a Document Type Definiation which contain root tag, Parent tag, child tag name

Here vendor are like apache ---etc.

Programmer Created → by using this structure we programmer created a xml file like ~~hibernate~~ hibernate configuration file, spring xml file, Struts xml file.

28-4-2015

DTD is used to create a structure of xml file.

→ a DTD is a file which defines a structure of xml document

→ In a DTD file, elements and attributes of elements are Created.

→ to create a DTD file the tags, & datatypes and the symbols & the keywords that are given by xml technology are used

→ if a DTD file is created then it means that one markup language of type xml is created.

→ a DTD file will be created by a service provider, to tell the client about How to Prepare xml document for Sending the data.

Example: 1

If address to be sent in the form of xml with addresses as root tag, address as Parent tag and hno, street, city as child tag then a related DTD file will be constructed like the following.

address.dtd

```
<!ELEMENT addresses (address+)>
<!ELEMENT address (hno, street, city)>
<!ELEMENT hno (#PCDATA)>
<!ELEMENT street (#PCDATA)>
<!ELEMENT city (#PCDATA)>
```

#PCDATA → Parsed character DATA

Reserved word | keyword indicator

→ While constructing DTD, two datatypes are used

1. CDATA
2. PCDATA

→ CDATA indicates unparsed character DATA and the PCDATA indicates Parsed character DATA.

→ If Datatypes is a CDATA then that element value will not verified by XML Parser.

→ In case of PCDATA, an XML Parser verifies the value of an element

→ If datatype is PCDATA type element then that element value

should not contain any special character except a white space and (_) @ underscore.

→ While creating a dtd we can use 3 occurrence operators also called **Cardinality operators**

① * → indicates that an element can occur 0 or more time

② + → indicates that an element can occur or repeated one or more times

③ ? → indicates that an element can occur 0 or 1 time.

→ In a DTD file, attributes can be defined for an element by using ~~declaration~~ <!ATTLIST> tag

<!ATTLIST element-name attribute-name attribute-constraint>

→ there are **3 attribute constraints** given by the xml technology.

① #REQUIRED

② #IMPLIED

③ #FIXED

→ REQUIRED means an attribute is compulsory for an element.

IMPLIED means an attribute is an optional.

FIXED means an attribute value is fixed.

for example

① <!ATTLIST address category #REQUIRED>

<address> → Invalid.

<address category = "family"> → valid.

<address category = "office"> → valid

② <!ATTLIST address category #IMPLIED>

<address> → valid

<address category = "family"> → valid.

<address category = "office"> → valid.

③ <!ATTLIST address category #FIXED "family">

<address> → Invalid.

<address category = "family"> → valid

<address category = "office"> → Invalid.

<!DOCTYPE → importing DTD file in xml document

In web-xml servlet container can add doctype for our web.xml.

→ <!DOCTYPE> element

→ when constructing xml document, in order to use the tags and there attributes that are defined in a DTD file, we need to import that DTD file into xml document. for importing we use <!DOCTYPE> element

→ when importing a DTD file, if it is a DTD ~~local~~ for local system then it is imported as SYSTEM DTD. and the if it is a DTD for all then imported as PUBLIC DTD.

→ the following is an xml document created by importing addressed.dtd

address_info.xml

for local system import as SYSTEM
for all system import as PUBLIC

root tag

```
<!DOCTYPE addresses SYSTEM
"-//Sathya Technologies//DTD addresses info//EN"
   owner of dtd        purpose of DTD      language code
"file:///c:/dtds/addresses.dtd">
                  ↳ name of dtd file
<addresses>
    <address category = "family">
        <hno> 11-1 </hno>
        <street> ameerpet </street>
        <city> hyd </city>
    </address>
    <adress category = "office">
        <hno> 1-11 </hno>
        <street> srnagar </street>
        <city> hyd </city>
    </address>
</addresses>
```

29-4-2015

Xml Schema Definition (XSD)

→ the following are the problems identified with DTD

① In DTD there are no operators given for specifing minimum and maximum number of occurrences for an element.

② a DTD has only two datatypes for creating the elements.

③ In DTD user defined datatypes can not be created

④ an XML Parser should follow different rules to check the validity of a DTD and to check the validity of a xml document. so a DTD will increase the burdon on a xml Parser.

→ To overcome the drawbacks of the DTD XML Technology introduced an alternate way of describing the structure of xml document called XSD.

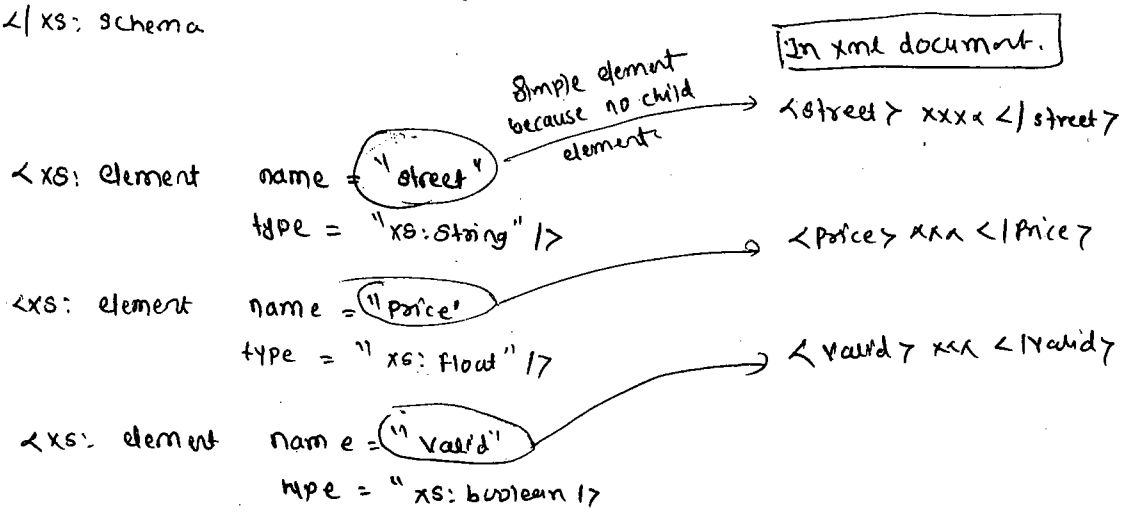
→ if we create an xsd file then it means we create a markup language of type xml.

→ In case of XSD we can specify minimum and maximum occurrances, we can create user defined datatypes and the XSD reduces the burdon of xml Parser.

<XS: Schema>

① Simple element → predefined data type
② Component element → user defined data type

</XS: Schema



<XS: element name = "address"
type = "addressType" />
↳ user defined datatype

→ to create a XSD file, the root element is <XS: Schema>.

→ In a XSD file we can describe two types of elements

① simple element.
② ~~Component~~ Compound element.

→ a simple elements means it is an element which consist data directly it cannot have child elements.

→ a ~~Component~~ Compound elements means it contains child elements, but not data directly.

→ either a simple element or ~~component~~ Compound element both are described in a XSD file using <XS: element> tag

→ for a simple element datatype will be predefined & for compound element a datatype is a userdefined.

for ex:

① a. simple element called a street can be described in XSD file like the following

```
<xs: element   name = "street"
               type = "xs: string" />
```

② In XSD file a compound element with address name "address" with child element "hno", "street", & "city" like a following

```
<xs: element   name = "address"
               type = "addressType" />   user define datatype.

<xs: complexType  name = "addressType">   (used to create user defined datatypes)
    <xs: sequence>
        <xs: element  name = "hno"  type = "xs: int" />
        <xs: element  name = "street"
                      type = "xs: string" />
        <xs: element  name = "city"
                      type = "xs: string" />
    </xs: sequence>
</xs: complexType>
```

⑤ A compound element with name "email" with sub element "from", "to", "subject", and "body' can be described in xsd like following

```
<xs: element  name = "email"
              type = "emailType" />

  <xs: complexType  name = "emailType">
    <xs: sequence>
      < xs: element name = "from" type = "xs: string" />
      < xs: element name = "to" type = "xs: string" />
      < xs: element name = "subject" type = "xs: string" />
      < xs: element name = "body" type = "xs: string" />
    </xs: sequence>
  </xs: complexType>
```

➪ **Describing an Attribute**

→ In an xsd file attributes are described by using <xs: attribute> tag

→ for example an attribute for an element address can be described in xsd file like the following.

↳ Next Page

```
<xs: element   name = "address"
               type = "addressType" />

<xs: complexType   name = "addressType" >
    <xs: sequence>
        ———————
        ———————
        ———————
    </xs: sequence>
    <xs: attribute   name = "category"
                     use = "required" />
</xs: complexType>
```

Note

① → <xs: attribute> tag is written or outside <xs: sequence> because in an element attributes doesnot have any sequence.

② → attributes can be written in any order in an element

30-4-2015

→ when describing an attribute, we can make the value of an attribute as a fixed.

→

```
<xs: attribute   name = "category"
                 type = "xs: string"
                 use = "required"
                 fixed = "family" />
```

→ we can set default value for an attribute, if it is described as optional.

```
<xs: attribute   name = "category"
                 type = "xs: string"
                 use = "optional"
                 default = "office" />
```

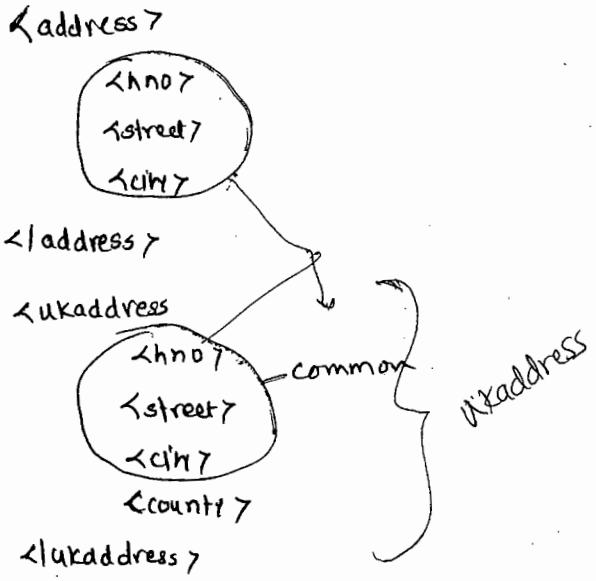
* Element Cardinality

→ In XSD file, element cardinality can be described by adding to attributes minOccurs and maxOccurs

①
```
<xs: element   name = "address"
               type = "addressType
            minOccurs = "3"
            maxOccurs = "5" />
```

②
```
<xs: element   name = "address"
               type = "addressType
            minOccurs = "0"
            maxOccurs = "unbounded" />
```

③
```
<xs: element   name = "address"
               type = "addressType"
            minOccurs = "0"
            maxOccurs = "1" />
```



* Inheriting the ComplexType

→ In XSD file, if there are two complexType and 2nd complexType contains elements of 1st complexType and also some extra elements then a 2nd complexType can be extended from 1st complexType.

→ for ex: we have complexType ukaddressType and the it has one extra element county than another complexType called addressType. then we can inherite ukaddressType from addressType like the following

```
<xs: element  name = "ukaddress"
              type = "ukaddressType" />

<xs: complexType name = "ukaddressType" >
    <xs: complexContent >
        <xs: extension  base = "addressType" >
            <xs: sequence >
```

```
                <xs: element  name = "county"
                              type = "xs: string" />
        <xs: sequence>
      <xs: extension>
    </xs: complexContent>
  </xs: complexType>
```

* Creating a SimpleType in XSD *

→ to Provide restriction on a Predefined datatype a simple type will be created.

→ a simple type can be added to the simple element only.

ex: 1 To Provide a restriction on the value of gender element, a simpleType can be created like the following

```
<xs: element  name = "gender"  type = "genderType" />
<xs: simpleType name = "genderType">
<xs: restriction  base = "xs: string">
        <xs: enumeration  value = "Male" />
        <xs: enumeration  value = "female" />
</xs: restriction>
</xs: simpleType>
```

<gender> abcd </gender> → invalid
<gender> male </gender> → invalid.
<gender> Male </gender> → valid.

ex2: To provide a restriction on length of a password element a simpleType created like the following

```
<xs: element name = "password"
                type = "passwordType" />

<xs: simpleType name = "passwordType">
    <xs: restriction base = "xs: string">
        <xs: length value = "8" />
    </xs: restriction>
</xs: simpleType>
```

<password> abcd </password> ⟶ Invalid

<password> weblogic </password> ⟶ valid

ex3: To provide a restriction on the value of age element a simpleType can be created like the following.

```
<xs: element name = "age"  type = "ageType" />
<xs: simpleType name = "ageType">
    <xs: restriction base = "xs: int">
        <xs: minInclusive value = "18" />
        <xs: maxInclusive value = "19" />
    </xs: restriction>
</xs: simpleType>
```

<age> 10 </age> ⟶ invalid

<age> 20 </age> ⟶ invalid

<age> 19 </age> ⟶ valid.

1-5-2015

XML Namespace

→ In xml terminology, a namespace is a nothing but a name in which element attribute and datatype that we created for constructing xml documents are stored

→ A namespace is same as to package name in java. In xml, multiple vendor creates namespace so there should be unique name id required for each name space to avoid the clash.

→ A recommendation is given for written a newspace URI. Use Domain name of company in namespace uri.

→ since, domain name of company are unique so namespace uri is also going unique.

→ When creating an xsd file. Its elements, attribute and datatype that are created in xsd file should be stored in one namespace. for this we add an attribute target namespace to the <schima> tag.

for example

addresses.xsd

```
<xs:schema targetNamespace : "HTTP://www.sathyaTech.com/schema/
                              addresses">
<xs:schema>
```

→ When creating an xsd file we need to use elements, attributes, datatypes that are given by xml technology.

→ xml technology has given Pre-defined namespace with all the tags, attributes, datatypes for creating a xsd files called http://www.w3.org/2001/XMLSchema.

→ When creating "xsd file" or "xml file", if we want to use elements, attributes or datatypes of a namespace then we need to import that namespace by using a keyword called xmlns.

→ xmlns keyword of xml is same as to import keyword of java.

→ By importing a name space we can added a Prefix also to that namespace.

for example.

addresses.xml
↓

To link xml and xsd file, xml technology give "schemaLocation" attribute.

↓

keyword Prefix

<xs: schema xmlns:xs = "http://www.w3.org/2001/schema/addresses"

targetNamespace = "http://www.sathyaTech.com/schema/addresses">

</xs: schema>

9/5/2015

→ without Prefix only one namespace written in xsd file

→ multiple namespaces can be written with Prefix in xml file.

→ while importing the namespace into an xml files, a man of namespace can be imported without a Prefix. we called the namespace as a default namespace in the xml file.

In xml namespaces are 3 types.

① default namespace.

② Source namespace.

③ Target namespace.

→ source namespace & Targetname space are used in xsd files.

→ In xsd file, the element of source namespace are ~~called~~ used for creating other elements (tag).

→ the newly created elements as tags are going to stored in targetnamespace.

→ when we are writting a xml file, we can import multiple namespace with xmlns keyword. but except for one name space for the remaining we need to add prefix.

Sample.xml

```
<addresses xmlns = "http://www.SATHYATECH.com/schema/addresses"
xmlns:p = "http://www.SATHYATECH.com/schema/Product">
        ↑
      Prefix
</addresses>
```

→ when constructing an xml file, we need to think XSD file with xml file, so at the time of Parsing, Parser checks for well ~~formness~~ formedness and validness of xsd file & then verifies wellformedness & validness of xml file.

→ To link XSD file to XML file, we need to add "Schema-location" attribute to the root tag of xml file & this attribute define in Pre-defined namespace of xml technology

SchemaLocation:

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

→ When writting xml file we need to import the above namespace

→ SchemaLocation attribute value contains 2 parts

1st Part = namespace URI

2nd Part = URL (Location of XSD file)

And both are separated by ~~what~~ white space. Each namespace contain 2 parts.

Ex

```
<addresses xmlns:"http://SATHYATECH.com/schema/addresses"
 xmlns:xsi="http://www.w3.org/2001/XMLschema-instance"
 xsi:schemaLocation="http://www.sathya.com/schema/addresse
                     file:///c:/schemas/addresses.xsd">
</addresses>
```

4-5-2015

Complete xsd file:

addresses.xsd

given by the technology

```
<xs:schema  xmlns:xs = "http://www.w3.org/2001/XMLSchema"
            targetNamespace = "http://www.sathyatech.com/Schema/addresses">
```

both are same.

```
<xs:element name = "addresses" type = "addressesType" />
<xs:complexType name = "addressesType" />
    <xs:sequence>
        <xs:element name = "address" type = "addressType" />
    </xs:sequence>
</xs:compleType>
<xs:ComplexType name = "addressType">
    <xs:sequence>
        <xs:element name = "hno" type = "xs:string" />
        <xs:element name = "street" type = "xs:string" />
        <xs:element name = "city" type = "xs:string" />
    </xs:sequence>
<xs:attribute name = "category" type = "xs:string" use = "Required" />
</xs:complexType>

</xs:schema>
```

Complete xml file.

addresses_info.xml

```
<addresses xmlns="http://www.sathyatech.com/schema/addresses"
           xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
           xsi:schemaLocation="http://www.sathyatech.com/schema/addresses
                               file:///C:/schemas/addresses.xsd">
    <address category="office">
        <hno>1-111</hno>
        <street>ameerpet</street>
        <city>hyd</city>
    </address>
    <address category="family">
        <hno>2-22</hno>
        <street>srnagar</street>
        <city>hyd</city>
    </address>
</addresses>
```

W3C

Dom specification

SAX Specification

JAX-P → contain classes & interfaces

javax.xml.parsers. → classes & interfaces related to dom & sax Parser

org.w3c.dom

org.xml.sax

---

① DocumentBuilderFactory (Abstract class)

② DocumentBuilder (Interface)

③

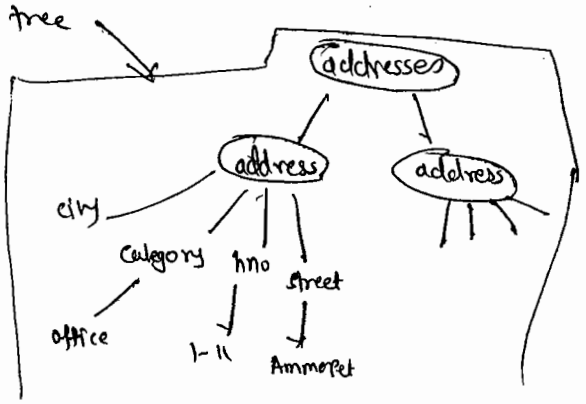
① SAXParserFactory (abstract)

② SAXParser (interface)

```
DocumentBuilderFactory factory = DocumentBuildrfactory.newInstance();
DocumentBuilder        db      = factory.newDocumentBuilder();
                       db.Parse(new file("addresses_info.xml"));
Document               d       = db.Parse("  ");
```

dom Parser

Document → Store in the form of tree



About Xml Parser

→ an xml Parser is also called as Xml Processor.

→ an xml Parser is a program who can check well formedness & validness of xsd file & xml file.

→ To create a xml Parser in a different languages, w3c has given two specification.

① dom specification

② SAX specification

→ Based on the dom & SAX specification Sun microsystm released Jax-P (Java API Xml Processing) API.

→ the implementation of a Interfaces of Jax-P api will be provided by vendors.

→ Jax-P API mainly contains 3 Packages

① Javax.xml.Parsers

② org.w3c.dom

③ org.xml.sax

→ Jax-P API contains not only the API for Parsers but also API for reading the content of xml file & and also writing the content to xml file.

→ the Dom & SAX Parser related API is given in Javax.xml.Parsers Package

→ DocumentBuilder interface is a Dom Parser & its object can be obtain through DocumentBuilderFactory (abs.class).

→ DocumentBuilder interface has Parse method & it will check wellformedness & validness of XSD & XML file

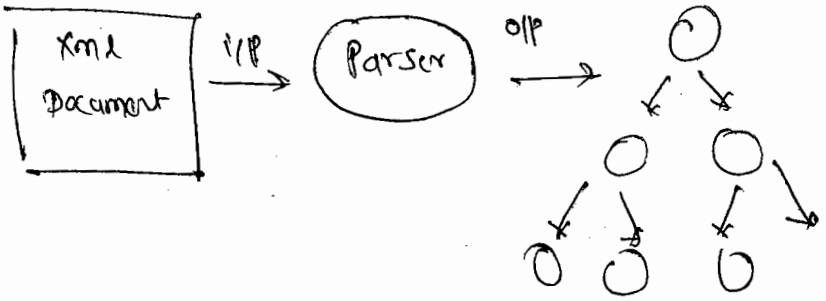
```
DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
DocumentBuilder builder = factory.newDocumentBuilder();
builder.parse(new File("addressed_info.xml"));
```

→ SAXParser is a interface is a sax Parser, and the we can obtain it's implement class Object using SAXParserfactory abstract class.

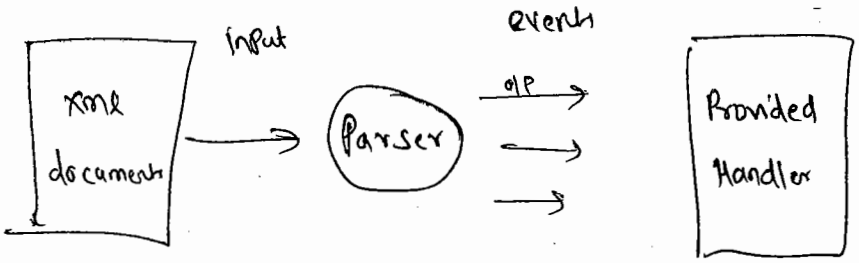
```
SAXParserFactory factory = SAXParserFactory.newInstance
SAXParser saxp = factory.newSAXparser();
saxp.Parse(new File("addresses_info.xml"));
```

Q

dom	SAX
① dom is a treebased Parser it means dom Parser converts an xml document into a free structure	① SAX is an event based Parser it means it will generate event if the xml document is valid (for each opening Parent tag & closing tag it generate event & for child also)
② Dom in Action	



SAX Operation Model



(2) Dom is suitable for small XML documents.

(2) SAX is suitable for large XML documents.

(3) Dom is used for both reading XML data & for writing into XML

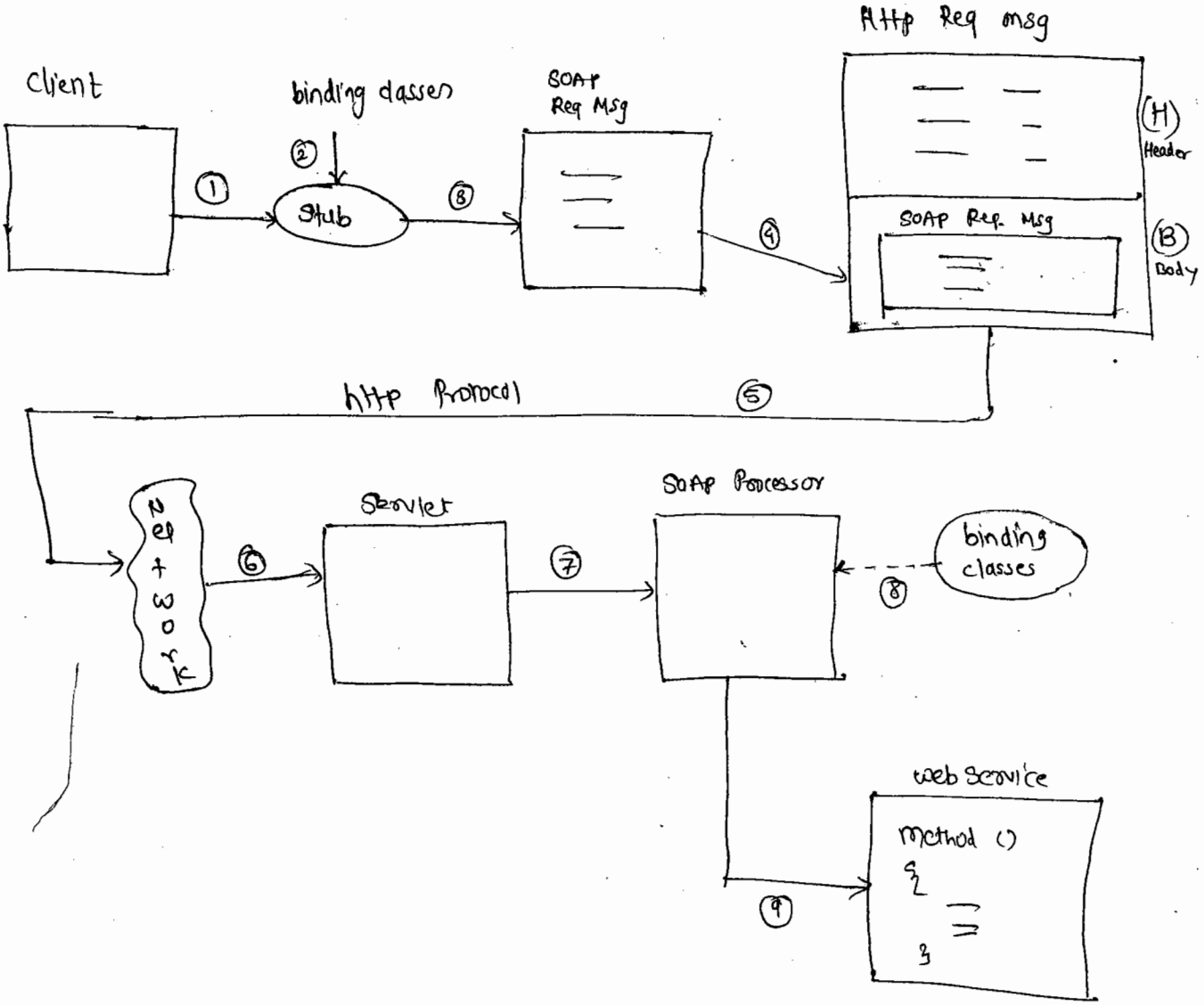
(3) SAX is only for reading for data from XML file

5-5-2015

SOAP Web Services

SOAP Based Webservices

Request flow Diagram



→ a Service Provider will shared a WSDL file of a web Service to a client, before the ~~actual~~ actual comm^n takes Place.

→ at client side by using some tool a WSDL file is passed and stub, binding classes for client side are generated.

Steps

① A client Appl^n calls method on Stub object.

②&③ a stub object takes the support of binding classes & converts a method call into a SOAP Request message.

④ a stub object creates a Http request & message & then inserts SOAP request message to body part & adds some Header information to Header part.

⑤ A stub transfers Http request message to http protocol across network to the server side.

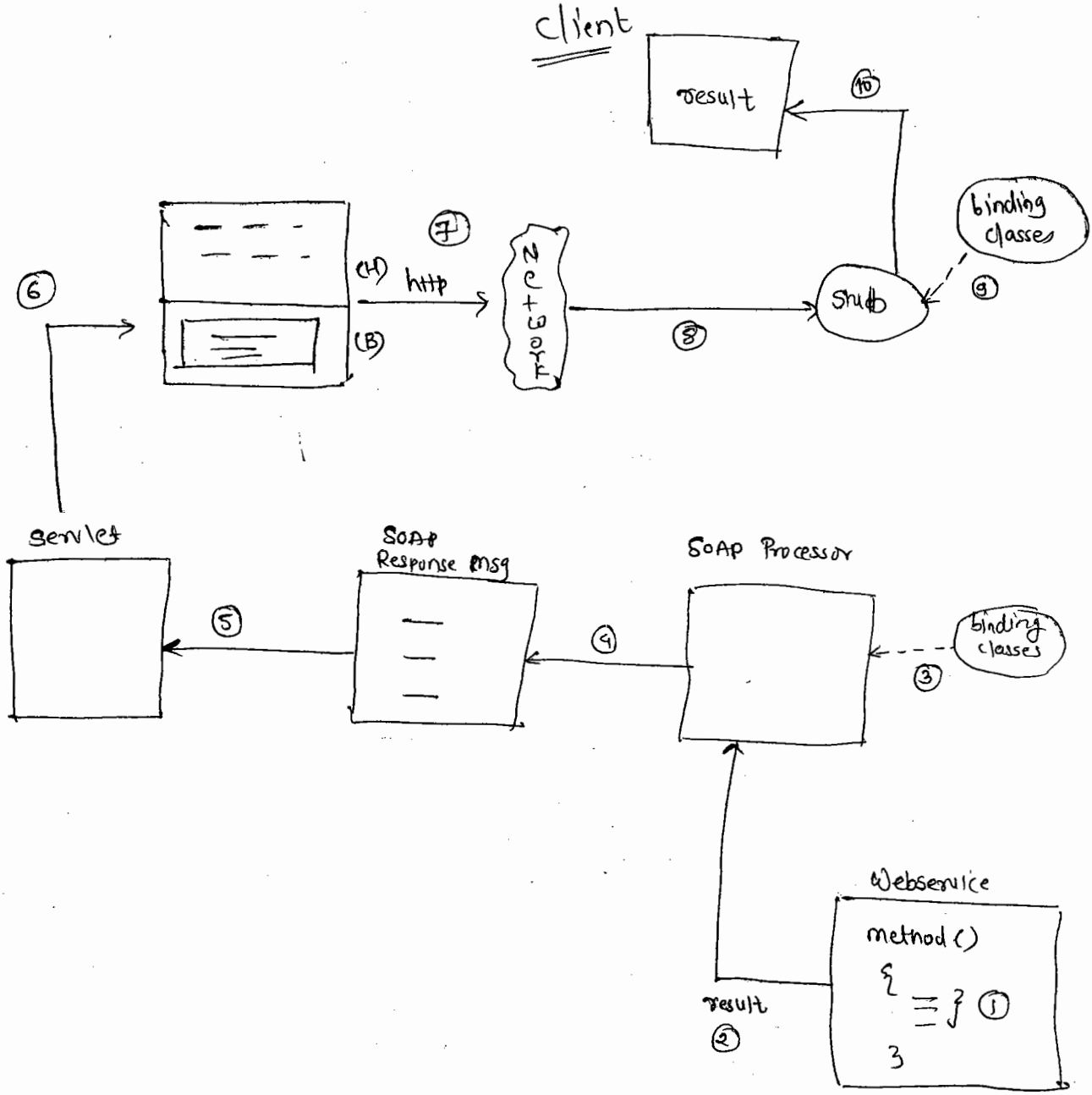
⑥ at server side, a servlet receives Http Request and reads Header part

⑦ Header part tells servlet that Body Parts contains SOAP request message. a servlet transfer a request to SOAP processor.

⑧ a SOAP Processor reads a SOAP ~~mess~~a request message from the Body part & then with the help Binding classes it converts a SOAP request message into method call.

⑨ finally a SOAP processor class calls the method of webservice

Response Flow Diagram



→ step

①&② → a method of a webservice is executed and it returns the result back to SOAP Processor

③&④ → SOAP Processor takes the Help of binding classes & converts result as "SOAP Response Message

⑤ SOAP Processor will sent SOAP Response message to a Servlet.

⑥ A Servlet creates HTTP Response message & then inserts SOAP response msg. to Body parts.

⑦ Using Http Protocol, Http Response message will be sent across network to the client side.

⑧ & ⑨ at a client side Stub object receives http response message and converts a SOAP Response message to a result, with the help of binding classes.

⑩ finally a Stub object returns result back to the client application.

6-5-2015

* WSDL

→ In web-services, a web service can developing one language and its client can be developed in another language.

→ In order to send information about a web service like methods in a web service, i/p & o/p and location of a web-service to the clients, a WSDL file will be generated.

→ A WSDL file contains 5 sections.

1. types section
2. Messages section
3. PortTypes section
4. binding section
5. service section

→ All the tags attributes that are required to constructing a WSDL file are given by WSI organization under one namespace "http://schemas.xmlsoap.org/wsdl/"

→ when constructing a WSDL file the above namespace will be imported into a WSDL file using xmlns keyword

(1) types Section :→

→ this section contains an xml schema.

→ In webservice operation (methods), either for i/p parameters or for return types of methods, if any equivalent complex types or simple types are needed in xml then they are defined or created under schema inserted in "type section".

→ If already pre-defined dataTypes are already available in xml for representing i/p parameters & return type of web service operation or methods. their types section does not contain any schema, It will be empty.

→ A schema (xsd) can be defined as inline or as outline.

→ Outline means. a schema can be defined at outside of WSDL file & it can be imported into types section of WSDL file.

7-5-2015

WSDL

<definitions xmlns = "http://schemas.xmlsoap.org/wsdl/"

xmlns: wsdl

<wsdl-definitions>

xmlns: xs = "http://www.w3.org/2001/XMLSchema">

Prefix

<types>

<xs: schema targetNamespace = "http://www.sathyatech.com/schema/types">

→ In a types section, to defined the schema, we need XmlSchema namespace to be imported.

→ if XmlSchema name space is imported within types section then elements of this name space can only used in types section. not in the Complete WSDL file. so in a WSDL file XmlSchema name space will be imported under root tag.

→ In a types section whatever the elements, complex type & simple types that are created will be stored in a target names space.

→ the elements of target names space are required for the message section so target names space is also imported with root tag with prefix tns

Sample.wsdl

```
<definitions  xmlns = "http://schemas.xmlsoap.org/wsdl/"
              xmlns:xs = "http://www.w3.org/2001/XMLSchema"
              xmlns:tns = "http://www.sathyatech.com/schema/types">

   <types>
      <xs:schema  targetNamespace = "http://www.sathytech.com/schema/types">
         —
         —
      </xs:schema>
   </types>
   —
</definitions>
```

```
public interface Calculator
{
   int add (int a, int b);
}

public class CalculatorImpl implements Calculator
{
   public int add (int a, int b)
   {
      =
   }
}
```

```
<message name = "addRequest">
    <Part name = "a" type = "xs: int" />
    <Part name = "b" type = "xs: int" />
</message>
<message name = "addResponse">
    <Part name = "Parameters" type = "xs: int" />
</message>
```

message section

```
Public double getData(Product p);

<message name = "getDataRequest">
    <Part name = "p" type = "tns: Product" />
</message>
<message name = "getDataResponse">
    <Part name = "Parameters" type = "xs: float" />
</message>
```

Message Section

→ In a Message Section two messages are created for each method (Operation) of a web service

→ a Service provider will tell a clients about how many Parameters should be send In a Soap request message when calling a webservice method and what type of output will come back in a SOAP Response.

→ In a message Signal, a message contains one or more Parts, where each Part Indicates one Parameter.

→ a message section can have multiple messages and each is identified with a name.

→ for example, if webservice method has 10 parameters then in message section a message contain 10 parts

→ In webservices if webservice method contains n parameters then n parameters values will come from the client as single SOAP request message.

for example

Service Endpoint Interface (SEI)

```
Public interface Calculator
{
    int add (int a, int b);
}

Public class CalculatorImpl implements Calculator
{
    public int add (int a, int b)
    {
        =
    }
}
```

Sample.wsdl

```
<definitions xmlns = "http://schemas.xmlsoap.org/wsdl/"
    xmlns:xs = "http://www.w3.org/2001/XMLSchema"
    xmlns:tns = "http://www.sathyatech.com/schema/types"
    targetNamespace = "http://www.sathyatech.com/schema/types">

<types/>
<message name = "addRequest">
    <part name = "a" type = "xs:int" />
    <part name = "b" type = "xs: int" />
</message>
<message name = "addResponse">
```

```
        <Part name = "Parameters" type = "xs: int" />
    </message>
    ----
</definitions>
```

Example 2

```
Public interfeice Demo
{
    Public double getData(Product P);
}

Public class DemoImpl implements Demo
{
    Public double getData(Product P)
    {
        =
    }
}
```

Sample.wsdl

```
<definitions xmlns = "http://schemas.xmlsoap.org/wsdl/"
    xmlns:xs = "http://www.w3.org/2001/XMLSchema"
    xmlns:tns = "http://www.sathyatech.com/schema/types"
    targetNamespace = "http://www.sathyatech.com/schema/types">
<types>
    <xs:schema targetNamespace = "http://www.sathyatech.com/schema/types">
        <xs:complexType name = "Product">
            <xs:sequence>
                <xs:element name = "pid" type = "xs:int" />
                <xs:element name = "pname" type = "xs:string" />
                <xs:element name = "price" type = "xs:float" />
            </xs:sequence>
        </xs:complexType>
    </xs:schema>
</types>
```

```
<message name = "getDataRequest">
    <Part name = "p" type = "tns:Product" />
</message>

<message name = "getDataResponse" >
    <Part name = "parameters" type = "xs:float" />
</message>
    ---
</definitions>
```

8-5-2015

WSDL

③ ~~PortType~~ ~~PortType~~ section
↓
contain one or more operation.

order of reading WSDL

- ~~PortType~~
- ~~PortType section~~
① - PortType section
  ↳ interface name
  → operation or method in interface.
  - I/P message
  - O/P message.
② message section.
③ type section

PortType Section

→ this section of WSDL file tells service Endpoint interface (SEI) and its method.

→ a PortType name is the interface name and the an operation name is a method name.

→ If an interface has 2 methods then under PortType operation will be repeated for 2 times.

→ every operation contains an I/P & O/P messages.

→ When WSDL file is given for us then first we need to refer or read PortType section. from PortType next we need see Message section. & then we need to see Schema under types section.

Example

```
Public interface Calculator
{
    Public int add (int a, int b);
    Public int multiply (int a, int b);
}

Public class CalculatorImpl implements Calculator
{
    ____
    ____
}
```

Sample.wsdl

```
<definitions xmlns = "http://schemas.xmlsoap.org
    xmlns:xs = "http://www.w3.org/2001
    xmlns:tns = "http://www.sathyatech.
    targetNamespace = "http://www.sathya

  <types>
  </types>
```

```
<message name = "addRequest">
    <Part name = "a" type = "xs: int" />
    <Part name = "b" type = "xs: int" />
</message>

<message name = "addResponse">
    <Part name = "parameters" type = "xs: int" />
</message>

<message name = "multiplyRequest">
    <Part name = "a" type = "xs: int" />
    <Part name = "b" type = "xs: int" />
</message>

<message name = "multiplyResponse">
    <Part name = "parameters" type = "xs: int" />
</message>

<PortType name = "Calculator">
    <operation name = "~~multiply~~ add">
        <input message = "tns: addRequest" />
        <output message = "tns: addResponse" />
    </operation>
    <operation name = "multiply">
        <input message = "tns: multiplyRequest" />
        <output message = "tns: multiplyResponse" />
    </operation>
</PortType>
---

</definitions>
```

Binding section → tells which protocol is used for transferring data

↓ Protocol
message format is used

Interview Question

Diff. rpc literal & Document literal

binding Section

→ A binding section tells the following 2 information

① which transfert Protocol is used for comm^n

② which message exchange format to be used for exchanging the i/p & o/p of a webservice operations.

→ In SOAP web services there are 4 message Exchanging format

① rpc by literal.

② rpc by encoded.

③ document by literal

④ document by encoded.

```
<binding name = "CalculatorBinding" type = "tns: Calculator">
    <soap : binding transport = "http://schemas.xmlsoap.org/soap/http" style="rpc" />
    <operation name = "add">
        <input>
            <soap: body use = "literal" />
        </input>
        <output >
            <soap: body use : "literal" />
        </output>
    </operation>
```

```
<operation name = "multiply">
    <input>
        <soap: body use: "literal" />
    </input>
    <output>
        <soap: body use = "literal" />
    </output>
</operation>
</binding>
```

Service Section

→ this section of WSDl file tells about the address location of a webservice in a network

→ a Service section contains one or more ports and each port refers binding section

```
<service name = "CalculatorService">
    <Port name = "CalculatorPort" binding = "tns: CalculatorBinding">
        <soap: address location = "http://localhost:2015/CalculatorApp" />
    </Port>
</service>
```

9-5-2015

* Simple Object Protocol:

→ SOAP is not a Protocol, it is kind of xml document.

→ SOAP is abbrivated as a Protocol, because it has its own Processing rules, & its own namespace.

→ In webservices comm", a client and web-service are developed in different language also in this case, the datatype and memory sizes of both languages will missmatch.

→ So, request and response are transfered in the form of SOAP messages.

→ At client side, a 'Stub' creates a soap request message. with the help of binding classes at client side, Similarly, a soap response message will be created by soap Processor with the help of binding classes at server side

SOAP message contains two part

① Header.

② Body

→ A SOAP message called an envelope and Header Part is optional and Body Part is mandatory.

Soap Message.

```
<Soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
<Soap:Header>          } optional
   ---
</Soap:Header>         }
<Soap:Body>            } mandatory
   --
</Soap:Body>           }
</Soap:Envelope>
```

→ when a clients calls a method of web service then a method call will be converted in xml, & then it will be inserted in Body Part of soap request message.

→ for one method call to another method call, xml will be different and input values are different in a soap Body. So the xml inserted in a soap Body is called an arbitrary xml.

→ In a soap body of soap request, an xml will be prepared with ip values according to message exchanging format defined in WSDL file.

→ A message exchanging format is combination of 'style' and 'use'

Style : rpc
document

Use : literal
encoded

So, message exchanging formats are

rpc/literal	document/literal
rpc/encoded	document/encoded

→ a difference bet rpc style & document style is, in rpc a method name will be used as root tag, in the business xml under soap body, in document style, a message name will be used as root tag for the business xml in soap body.

→ a difference bet literal & encoded use is, in case of literal only values are transferred in business xml, & if it

11-5-2015

- encoded values with types are transferred in business xml.

for example, if we have a method in webService hello with input as string Parameter then in soap request message its body Part contain business xml for different message exchanging format like the following

① rpc|literal

```
<soap: Body>
    <hello>
        <argo> abcd </argo>
    </hello>
</soap: Body>
```

← Business xml or arbitrary xml

② rpc|encoded.

```
<soap: Body>
    <hello>
        <argo type = "xs:string"> abcd </argo>
    </hello>
</soap: Body>
```

③

↳ next Page

③ document | literal:

```
<soap:Body>
    <ns2:helloRequest  xmlns:ns2="http://www.sathyatech.com/schema1/types">
        <txt>abcd</txt>
    </ns2:helloRequest>
</soap:Body>
```

(name, message, part name)

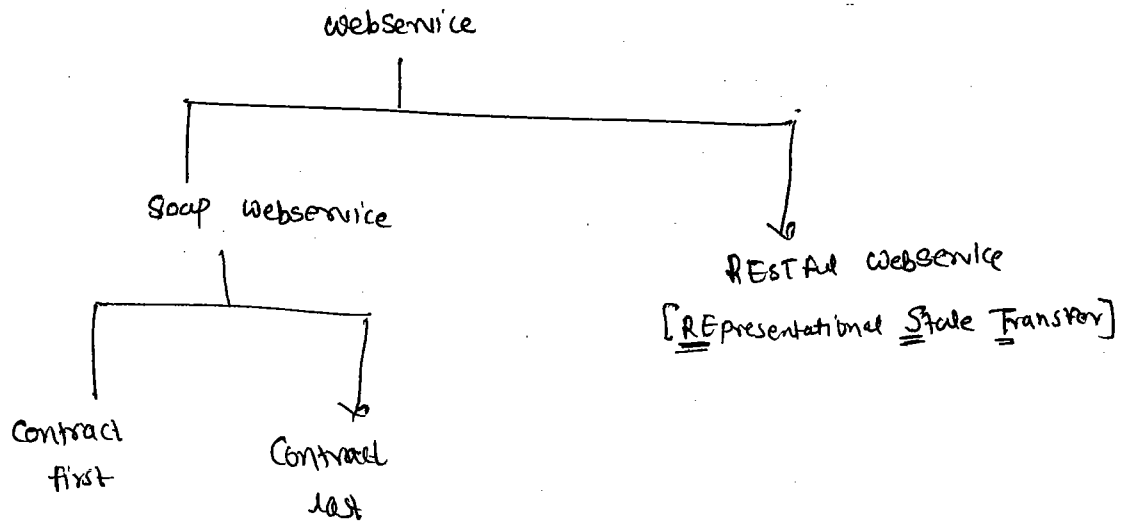
④ document | encoded

```
<soap:Body>
    <ns2:helloRequest  xmlns:ns2="http://www.Sathyatech.Com/schema1/types">
        <txt type="xs:string">abcd</txt>
    </ns2:helloRequest>
</soap:Body>
```

Contractfirst → WSDL creates first.
→ before write code prepare WSDL file
→ Based on WSDL, web service prepare.

Contractlast (service first)

Q How many types of webservices are there?



→ In Contract first, WSDL file (contract) will be created first & Based on that webservice will be created.

→ In Contract Last approach, a webservice class will be created first & Based on that a WSDL file will be created.

JAX-RPC API	JAX-RPC SI
	Apache AXIS
	oracle weblogic
	JBoss
	IBM websphere

JAX-WS API	JAX-WS R1
	Apache AXIS2
	Apache CXF
	Glassfish METRO
	Oracle Weblogic
	IBM Websphere
	JBoss

JAX-RPC

→ WS-I organisation has released ~~three~~ two specification for creating soap webservices.

1. Basic Profile Version 1.0
2. Basic Profile Version 1.1

→ for BP Version 1.0 specification, sun microsystem released JAX-RPC API

→ for BP Version 1.1, Sun microsystem released JAX-WS API.

→ JAX-RPC api & JAX-WS API's contains api for developing webservice classes & also for developing client application

→ JAX-RPC & JAX-WS API's contain interfaces & implementations will be provided by vendors.

JAX-RPC API

- JAX-RPC SI (SUN) ✓
- Apache AXIS
- Oracle weblogic
- JBoss
- IBM websphere

} Vendors Provided implementation class for JAX-RPC API

JAX-WS API

- JAX-WS RI (SUN) ✓
- ~~Apache~~ Apache AXIS2 ✓
- Apache CXF
- Glassfish METRO ✓
- Oracle Weblogic
- IBM Websphere
- JBoss

→ To use Sun microsystem provided JAX-RPC SI for creating a soap based webservice, we need to download &

Install ~~JWSDP~~ JWSDP-2.0 (Java web services developer pack).

→ JWSDP-2.0 is compatible for ~~JDK~~ Java 1.5 & 1.6.

12-5-2015

WS Compile tool generate WSDL file. JAX-RPC RI

```
Public interface extends Remote        → marker interface in java.rmi package
{
   Public String xxx (-) throws RemoteException;
}
```

Config.xml

WS Compile → binding classes & wsdl

ws compile

-gen: server

-gen: client

JAX-RPC Example

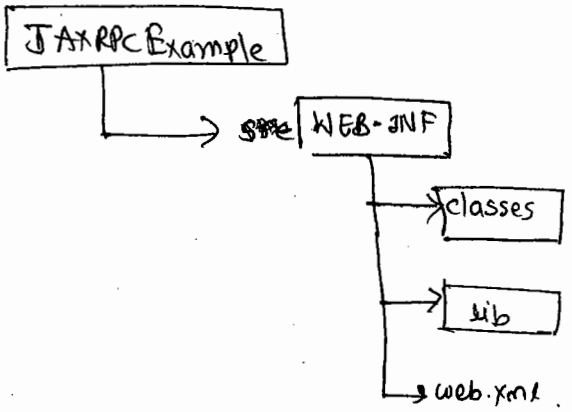
→ We are going to developeed a webservice with name Stock which returns a stock price for the stock id

→ To develop a webservice using JAX-RPC API, we must create ~~one~~ an interface & its implementation class.

→ To make our webservice as accessible across network for all clients, we need to store our webservice files in a web appl^n structure & we need to deploy our webservice in a server.

→ the following are the steps for creating webservice with JAX-RPC API using JAX-RPC SI & implementation given by SUN.

Step ① :- create a directory structure



Step 2 : Define interface & implementation class and store them under classes Directory.

```
// SEI Interface
// Stock.java

Package com.satrya.jaxrpc.webservice.
import java.rmi.Remote;
Import java.rmi.RemoteException;
Public Interface Stock extends Remote
{
    Public double getStockPrice (String stockId) throws
    RemoteException;
}
```

```
// Implementation class
// StockImpl.java

Package com.sathya.jaxrpc.webservice;

Public class StockImpl implements Stock
  {
      Public double getStockPrice(String stockId)
      {
         if ("S1234".equals(stockId))
            return 123.45;
         else if ("S4321".equals(stockId))
            return 321.54;
         else
            return 0;
      }
  }
```

c:\> Path = c:\Program Files\Java\jdk1.5.0_10\bin

c:\> cd JAX-RPCExample

↓

WEB-INF

Step3 Compile interface & implementation class

c:\> Path = c:\Program files\Java\jdk1.5.0_10\bin

JAX-RPCExample → WEB-INF → sta classes.

e:\JAX-RPCExample\WEB-INF\classes > javac -d . *.java

Step 4 :→ Create a Config.xml and the ~~Stocks~~ store it in under WEB-INF directory

→ ~~Conf~~ Config.xml will be used by WSCompile tool for generating binding classes and WSDL file for a server side

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration xmlns="http://java.sun.com/xml/ns/jax-rpc/ri/config">
    <service name="StockService"
        targetNamespace="http://sathya.org/stock"
        typeNamespace="http://sathya.org/types"
        packageName="com.sathya.jaxrpc.webservice.binding">
        <interface name="com.sathya.jaxrpc.webservice.Stock"
            servantName="com.sathya.jaxrpc.webservice.StockImpl"/>
    </service>
</configuration>
```

~~this namespace will be used as a~~

→ this namespace will be used as a ~~name~~ targetNamespace in definitions ~~section~~ tag of WSDL file

→ this namespace is used to store datatypes created in schema under types section wsdl file.

→ In this package the binding classes generated by wscompile tool will be stored

Step 5 → To generate a Binding classes and the wsdl file with wscompile tool, we need to pass the following option for the wscompile tool

-gen:server → (it tells to) generate Binding classes at Server side.

-d WEB-INF/classes :→ it tells the to store Binding classes in classes directory

-cp WEB-INF/classes :→ It will set the classpath to the service interface & implementation class stored in classes Directory.

-keep :→ it tells to keep source code of the binding classes

~~-model~~ →

-model model.xml.gz → it tells to store the internal structure of wsdl in a model file model.xml.gz

→ We need to type the wsdl compile at root directory

C:\JAX-RPCExample> set Path = %Path%; C:\sun\jwsdp-2.0\jaxrpc\bin; C:\sun\jwsdp-2.0\jwsdp-shared\bin

C:\JAX-RPCExample> wscompile -gen:server -d WEB-INF/classes -cp WEB-INF/classes -keep -model model.xml.gz WEB-INF/config.xml

13-5-2015

Step 6

→ copy StockService.wsdl from classes directory to WEB-INF directory and the model.xml.gz from root directory to WEB-INF directory.

Step 7

→ create a jaxrpc-ri.xml for configuring a webservice & store this file under WEB-INF directory.

jaxrpc-ri.xml

```
<?xml version = "1.0" encoding = "UTF-8" ?>
<webServices
    xmlns = "http://java.sun.com/xml/ns/jax-rpc/ri/dd"
    version = "1.0"
    targetNamespaceBase = "http://sathya.org/stock"
    typeNamespaceBase = "http://sathya.org/types"
    urlPatternBase = "/stock">

    <endpoint
        name = "MyStock"
        displayName = "Stock Service"
        description = "/WEB-INF/StockService.wsdl"
        interface = "com.sathya.jaxrpc.webservice.Stock"
        implementation = "com.sathya.jaxrpc.webservice.StockImpl"
        model = "/WEB-INF/model.xml.gz" />

    <endpointMapping
        endpointName = "MyStock"
        urlPattern = "/stock" />
</webservices>
```

Step 8

→ Create a web.xml with session-timeout & store this file under WEB-INF

```
<web-app>
    <session-config>
        <session-timeout> 30 </session-timeout>
    </session-config>
</web-app>
```

Step 9

→ add following list of jars to the lib folder

1. activation
2. FastInfoset
3. javax-xml-stream-1-0-1
4. jaxp-api
5. jaxrpc-api
6. jaxrpc-impl
7. jaxrpc-spi
8. mail
9. saaj-api
10. saaj-impl
11. xercesimpl
12. xmlsec

Step 10

→ Create a war file for the webservice application & then create final deployable war file using wsdeploy tool

```
c:\JAX-RPCExample7> jar cvf sample.war *.*

c:\JAX-RPCExample7> wsdeploy -o mystocks.war sample.war
```

Step: 11

→ Deploy MyStocks.war in webapps directory of tomcat6

→ Note → when installing the tomcat 6 Server we need to select a compatible jre like ~~jre~~ jre5/jre6

Step 12

→ Start the server and then type the following request in address bar, to know the webservice is deployed successfully or not.

http://localhost:2017/MyStocks/stock

MyStocks → war file name.

stock → url Pattern

Creating a Client for the above JAX-RPC WebService

Step1

→ create a directory in c drive with name JAX-RPCClient

C:
└→ JAX-RPCClient.

Step:2 create a config.xml & save it in client folder

→ Config.xml

```
<configuration xmlns="http://java.sun.com/xml/ns/jax-rpc/ri/config">
<wsdl location="http://localhost:2017/MyStocks/stock?WSDL" PackageName
="com.sathya.JaxRpc.client">
</wsdl>
</configuration>
```

Step:3

Generate binding classes and Stub using wscompile tool

e:\JAX-RPCclient> wscompile -gen:client -keep config.xml

14-5-15

Step 4:

Create a client appln & store it in ~~client~~ JAXRPCClient folder

```
//client.java

import com.sathya.jaxrpc.client.*;
import javax.xml.rpc.*;
import java.rmi.*;
class Client
  {
    main throws ServiceException, RemoteException
    {
      StockService ss = new StockService_Impl();
      Stock s = ss.getStockPort();
      double p = s.getStockPrice("SJ234");
      System.out.println("price=" +p);
    }
  }
```

Step 5 → add all the Jax-Rpc related jar to the class path.

→ Compile & execute client.java

e:\JAX-RPCclient> javac client.java

C:\JAX-RPCclient> java client

MEP's

① Synchronous request/response Pattern → RPC supported + WS
② Asynchronous → WS
③ One-way message. → RPC + WS supported

JAX-WS API

differences bet^n JAX-RPC & JAX-WS API

JAX-RPC	JAX-WS
① To create a web service an interface and implementation class both are required or Compulsory	① To create a webService, interface is optional
② if a webservice method name is changed by the service provider then we need to make the changes to client side Application also, by generating new Binding class	② In JAX-WS API, we can provide alias names to method & the alias names are given to client so if a method name is changed in a webService then there are no changes needed at client side.
③ JAX-RPC API supports only synchronous reques/ryponse & one way messaging Pattern.	④ JAX-WS API Supports synchronous request/response, Asynchronous request/response & one way messaging Pattern.
④ We don't have annotation in Jax-rpc API.	⑤ We have annotation Support in Jax-WS API

⑤ In JAX-RPC API we can not hide any interface method from the clients. it means all methods of interface are exposed as operations of a webservice to the clients.	⑥ In JAX-WS API we can hide some methods of interface from the clients.

IMP

* Basic Annotations of JAX-WS API *

→
- @ WebService } compulsory
- @ webmethod
- @ webParam
- @ webResult

} optional

→ the above annotation are given under javax.jws package if it is a part of java api.

→ for JAX-WS api, Sun microsystems given a reference implementation and it is added to the java software only from java 6

① @WebService

→ this annotation is the compulsory annotation on top webservice interface & also webservice class.

→ the elements of @ WebService annotation are

① targetNamespace

→ It is used to pass a name space uri in which we want to store elements that are created in wsdl file & xml datatype that are created in wsdl file.

→ if we dont add this element then package name

of a service is in reverse order will be taken as Namespace.uri.

② PortName:

→ To this element we can pass a portName to be used in service section of wsdl file.

→ If this element is not added then a tool which generating wsdl file will by default attach some name for the Port under service section.

15-5-15

③ ServiceName:-

→ To this element we pass a name that we want for service section of a wsdl file.

```
@WebService(serviceName = "StockService")
```

④ name: → interface name

→ to this element we pass a name that we want provide for portType section of wsdl file.

→ If we dont pass this element then by default interface name will be given to the portType section

⑤ endpointInterface:

→ to this element we need to pass fully qualified interface name if a webservice class is implementing an interface

→ this element will be added to @WebService annotation when an annotation is added on top of class

@WebService (endpointInterface = "com.sathya.jaxws.Stock")

② @WebMethod

→ this annotation can be added on top of methods of webservice interface or on top methods of webservice class.

→ this annotation is used in following 2 cases

① when we want to provid alias names of webservice method as operation names in wsdl file.

② when we don't want to include a webservice method as an operation in a wsdl file.

→ @WebMethod annotation has 2 elements

① operation name :→ to this element we pass alias name

② exclude :→ default value is false, we will set as true if we want to exclude a method as an operation

```
@WebService
Public interface Stock
{
    @WebMethod (operationName = "m1")
    Public double getStockPrice (String id);

    @WebMethod (exclude = true)
    Public String getStockName (String id);
}
```

③ WebParam

→ this annotation is applicable for parameters of webmethods

→ when a request comes from a client in soap Body different tags will be used in rpc & document style for sending parameter values.

→ if we want to read a parameter value from soap Body from same tag in both RPC & document style then we use @WebParam annotation for a parameter.

```
@WebService
Public interface Stock
{
    @WebMethod
    Public double getStockPrice(
        @WebParam (name = "input")
        String stockId);
}
```

④ WebResult

→ this annotation is applicable for webservice methods.

→ when a webservice method is returns a result, then the result will be send in soap response with the tag .return

→ If we want to send the response of a webservice method to the client with a some other tag name than return then we add @WebResult annotation on top of the method.

Example

```
@WebService
Public Interface Stock
{
    @WebMethod
    @WebResult (name = "output")
    Public double getStockPrice (String stockId);
}
```

⑤ @SOAPBinding

→ this annotation is used to set a message exchanging format for a webservice

→ this annotation is applicable for a webservice interface or a webservice class.

→ this annotation has 2 element

① Style ② Use.

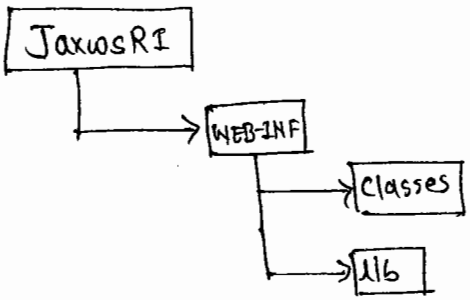
```
@WebService
@SOAPBinding (style= SOAPBinding.Style.RPC,
              use= SOAPBinding.Use.LITERAL)
Public interface Stock
{
    =
}
```

→ In JAX-WS webservices, a webservice class should follow the below 3 rules.

① class must be public
② class must ~~not~~ contain a default constructor.
③ class must be in package.

Creating a JAX-WS Application Using Reference Implem^{n} (RI) of sun

Step ① create a directory structure.



Step ② define Interface and Implementation class of webservice.

```
// Book.java
// SEI
Package com.sathya.jaxws;
import javax.jws.WebService;
import javax.jws.WebMethod;
```

```
@WebService (name = "Book", targetNamespace = "http://www.sathya.org/book")
```

<portType = "Book"> ~~<portType = "Book">~~ ← in wsdl file.

```
Public interface Book
{
    @WebMethod (operationName = "bookPrice")   ← alias name for method getBookPrice.
    Public double getBookPrice (String isbn);
}
```

// BookService.java

// Implementation class

```
Package com.sathya.jaxws;
import javax.jws.WebService;
@WebService (endpointInterface = "com.sathya.jaxws.Book",
             serviceName = "BookService",
             portName = "BookPort")
```

<service name = "BookService">
<port name = "BookPort">

alias name for class

```
Public class BookService implements Book
{
    Public double getBookPrice (String isbn)
    {
        if ("IS101".equalsIgnoreCase(isbn))
            return 230.34;
        else if ("IS102".equalsIgnoreCase(isbn))
            return 499.56;
        else
            return 0;
    }
};
```

16-5-2015

classes

lib

→ sun-jaxws.xml ⇒ used for end point configuration

→ web.xml → contain listner & servlet

Step 3

→ create sun-jaxws.xml & web.xml and then stores both files in WEB-INF directory.

→ sun-jaxws.xml contains end point configuration of a web service.

sun-jaxws.xml

```
<endpoints
    xmlns = "http://java.sun.com/xml/ns/jax-ws/ri/runtime" version="2.0">

    <endpoint
        name = "provider"
        implementation = "com.sathya.jaxws.BookService"
        url-pattern = "/books" />

</endpoints>
```

→ In web.xml, we need to configure a listner and a servlet class

→ a listner class reads end point configuration from sun-jaxws.xml and then it will store the information ~~end po~~ information of end point in ~~context~~ ServletContext object with key as end point name.

→ a listner class will do the above work at the deployement time of the web appln

→ a Servlet class traps the request that comes from client and reads end point configuration information from Context object and then it checks request url pattern is matched with end point url pattern or not.

→ it matched then allows the request, otherwise throws an exception

→ to read end point data from context, a servlet alias name & endpoint name should be same.

web.xml

```
<web-app>
    <listner>
        <listner-class> com.sun.xml.ws.transport.http.servlet.WSServletContextListner </l-s>
    </listner>
    <servlet>
        <servlet-name> Provider </servlet-name>
        <servlet-class> com.sun.xml.ws.transport.http.servlet.WSServlet </servlet-class>
        <load-on-startup> 1 </l-o-s>
    </servlet>
    <servlet-mapping>
        <servlet-name> Provider </servlet-name>
        <url-pattern> /* </url-pattern>
    </servlet-mapping>
    <session-config>
        <session-timeout> 80 </session-timeout>
    </session-config>
</web-app>
```

Step 4

jwsdp-2.0 ← download

→ copy the following jars to the lib folder of the web application.

- activation
- FastInfoset
- http
- jaxb-api
- jaxb-impl
- jaxb-xjc
- jaxws-api
- jaxws-rt
- jaxws-tools
- saaj-api
- saaj-impl
- Stax-ex
- streambuffer

Step 5

→ compile interface & implementation class of webservice

```
c:\JaxwsRI\WEB-INF\classes> javac -d . *.java
```

Step 6

→ generate the binding classes for server side using wsgen tool

```
c:\JaxwsRI\WEB-INF\classes> wsgen -cp . -keep com.sathya.jaxws.BookService
                                  ↑
                                space
```

Note

→ wsgen tool will stored the generated binding classes in com.sathya.jaxws.jaxws Package.

Step 7

→ create a war file for the application

e:\JaxwsRI>jar cvf RI.war . ← represent current direct

→ deploy war file in a Tomcat server & start the server.

Note
→ to see the wsdl file type the following url in address bar.

http://localhost:2015/RI/books?wsdl

Creating a client appl" for above ~~App~~ web service.

step① create a folder with name JaxwsRIClient.

② generate a binding classes for client side using wsimport tool.

e:\JaxwsRIclient>wsimport -p com.satya.client -keep http://local
host:2015/RI/books?wsdl

↓ to store the binding classes for client

Parsing WSDL---

Generating code---

compiling code---

③ create a client program like the following & save it under client directory

Client.Java

```
Import com.sathya.client.BookService;
import com.sathya.client.Book;

class Client
{
  main
  {
    BookService service = new BookService();
    Book b = service.getBookPort();
    double d = b.bookPrice ("IS101");
    SOPln(d);
  }
}
```

e:\JaxwsRIClient>Javac Client.Java

e:\JaxwsRIClient>Java Client

230.34

18-5-2015

Q You created webservice, How you verified that its working properly or not?

→ tool is there SOAP UI
& for Glassfish server there is tester Page.

Creating a Webservice in Netbeans 8.x

→ with netbeans Id automatically Glassfish-4.x and Tomcat 8.x servers will be automatically installed

→ Glassfish is an application server, it comes with a tomcat as a default web container.

→ the Port No. of that tomcat is 8080 and it will get a clash with oracle http service so we need to change the tomcat Container portno. in a glassfish like the following.

c:\Program files\glassfish-4.1\glassfish\domains\domain1\config\domain.xml and change <network-listener> port 8080 to 2020.

→ in a Glassfish Server, there is a built-in webservices engine given by Glassfish community called metro & it will generate binding classes for webservice, wsd files & the xml files. automatically. so it will be easy to create a webservice. to deploy on a glassfish.

→ a webservices engine in glassfish, also creates a webservice tester Page for testing the operations of a webservice, by

Without Preparing Client Application.

Step ① Start Netbeans IDE → File Menu → New Project → Project Name → MoveWebservice → Next → Server → Glassfish → next → finish

Step ② Expand Project → expand Web Page → delete index.html.

③ Right click on Source Packages → Other → Select Java Categories → Java Interface → enter name → Movie. Package com.sathya.webservice → finish

Movie.java

```
Package com.sathya.webservice;
Import java.util.List;
Import javax.jws.webservice;
@WebService
    Public interface Movie
    {
        Public List getMovies(int year);
    }
```

Right click on Source Package → new → java class → class name = MovieLibrary, package = com.sathya.webservice → finish.

MovieLibrary.java

```
Package com.sathya.webservice;
```

```
import java.util.ArrayList;
       java.util.List
       javax.jws.WebService;

@WebService (endPointInterface = "com.sathya.webservice
                                  .Movie)

Public class MovieLibrary implements Movie
 {
   Public List getMovies (int year)
   {
     List list = new ArrayList();
     if (year = 2015)
      {
        list.add ("LION");
        list.add ("GANGA");
        list.add ("GABBAR");
      }
     else
     {
       list.add ("sorry, year should be 2015);
     }
     return list;
   }
 }
```

step → Right click on Project name → deploy

→ to see the wsdl file of the above webservice we need to type the following url in addressbar

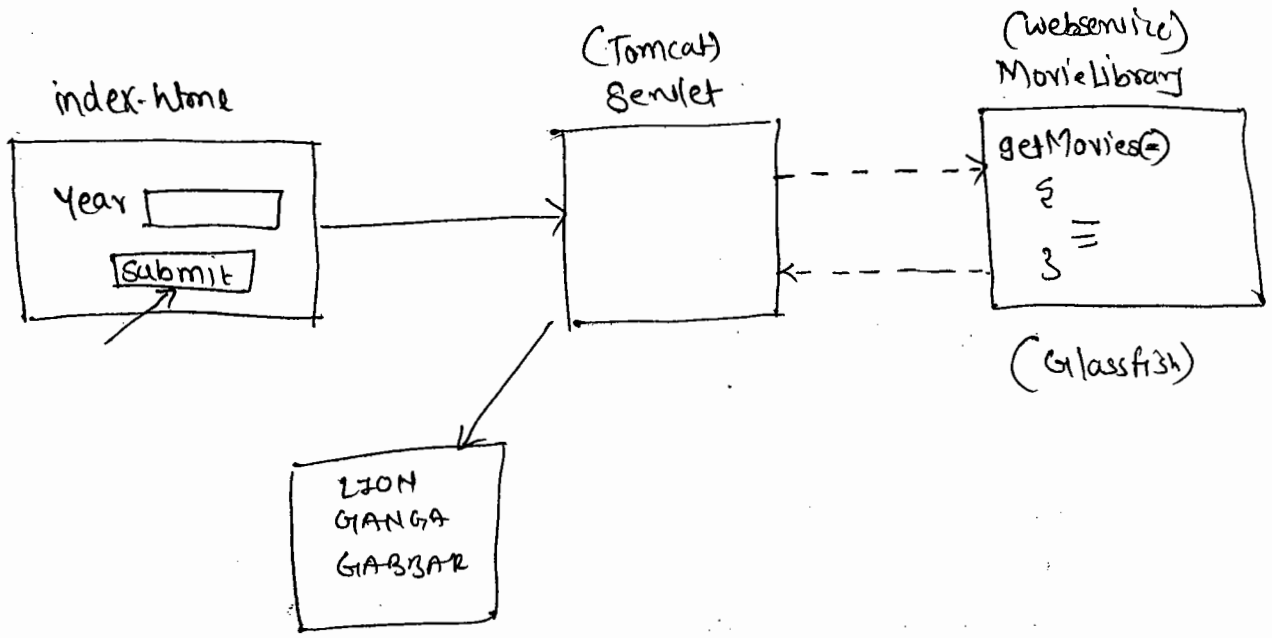
http://localhost:2020/MovieWebService/MovieLibraryService?WSDL

Note:- ~~beter~~ Web services engine in glassfish Server will create a Url Pattern for a web service by using the formula [Webservice class name + Service]

step → Open tester page created by Glassfish on browser like the following

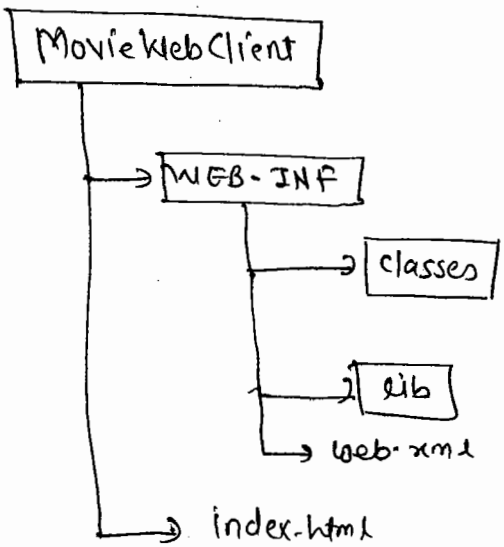
http://localhost:2020/MovieWebService/MovieLibraryService?Tester

Creating a Servlet as a Client for the above WebService.



→ Here a Webapplication with Servlet is a client for webservice

→ Servlet is going to run on tomcat and it is going to call a webservice running on Glassfish Server



Glassfish Server
→ localhost : 4848 / common / index.jsf

19-5-2015

Step 1

```
<!-- index.html -->
<center>
  <form action = "Sout1">
    Year: <input type = text name = "year"> <br>
          <input type = submit value = "submit">
  </form>
</center>
```

Step 2

Generating a Binding classes for a WSDL file like the following

D:\MovieWebClient\WEB-INF\classes > wsimport -p com.sathya.client -keep

http://localhost:2020/MovieWebService/MovieLibraryService?WSDL ↵

Parsing WSDL....

Generating code...

Compiling...

(Binding classes generated at client side)

Step 3

~~Servlet~~

Create Servlet1.java and store it in classes directory

```
// Servlet1.java

Package Com.sathya.servlet.

Import javax.servlet.*;
       java.io.*;
       java.util.*;
       Com.sathya.client.*;

Public class Servlet1 extends GenericServlet
{
   Public void service (
   {
      // read input value

   int year = Integer.parseInt (request.getParameter ("year").trim());

   // call getMovies() of webservice.

   MovieLibraryService service = new MovieLibraryService();

      Movie movie = service.getMovieLibraryPort();

      List list = movie.getMovies (year);

   // Set MIME type.

   response.setContentType ("text/html");

   PrintWriter out = response.getWriter();
```

```
Iterator it = list.iterator();
while (it.hasNext())
{
    out.println(it.next());
    out.println("<br>");
}
out.close
}
}
```

Step4

```
|WEB-INF|classes> javac -d . Servlet.java
```

Step5

```
<web-app>
  <servlet>
    <servlet-name>sone</servlet-name>
    <servlet-class>com.sathya.servlet.Servlet1</s-c>
  </servlet>
  <servlet-mapping>
    <servlet-name>sone</servlet-name>
    <url-pattern>/srv1</u-p>
  </s-m>
</web-app>
```

Step6 → copy MovieWebClient directory to Tomcat/webapps/ & then start the tomcat server.

→ open the browser & type the following request

~~Http://local~~

Http://localhost:2015/MovieWebClient

Creating a Console Appln for MovieWebService using NetBeans IDE

Step1 file Menu → New Project → Select Java → Java Appln → Next

→ enter Project Name = JavaApplication1 & unchecked create Main class check Box. → finish.

Step2 Right click on project JavaApplication1 → new → other → select the WebServices at left side → webservice client at right side → next → select wsdl and enter the url

http://localhost:2020/MovieWebService/MovieLibraryService?wsdl

→ enter Package name P1 → finish

Step3 Copy P1 Package from generated Source to Source Packages

④ → Right click on Project Name → new → java class → enter class Name = Main → finish.

Next Page ↳

```
Public class Main
{
  private static java.util.List<java.lang.Object> getMovies(int arg0)
  {
    p1.MovieLibraryService service = new p1.MovieLibraryService();
    p1.MovieLibraryPort port = service.getMovieLibraryPort();
    return port.getMovies(arg0);
  }

  Public static void main (String args[])
  {
    List list = getMovies(0015);
    Iterator it = list.iterator();
    while(it.hasNext())
    {
      sop(it.next());
    }
  }
}
```

Note → In the above main class we no need of writing getMovies() method manually.

→ we can Drag & Drop getMovies() method into main class

→ expand webService references → expand MoviesLibraryService → expand MovieLibraryPort → Drag & Drop getMovies() method to the main class.

Step 5 Right click on main class Select run file

20-5-2015

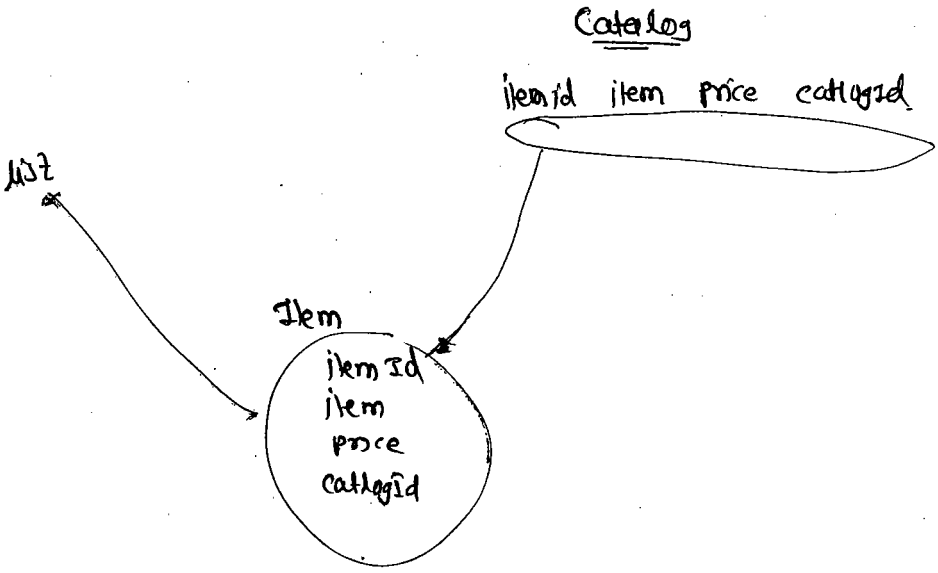
→ Creating a WebService in a eclipse IDE

Using Reference Implementation (RI) given by sun.

Catalog (I)

getCatalogItems (int catalogId)

CatalogService (c)



→ In the following example we are creating a operation getCatlogItems() It will take input as a catalogId & returns a list of items objects, by reading the data from database

Step1 → new → Dynamic web Project → ProjectName = CatalogApp

Module Version = 2.5 → finish

Step2 → Right click on src → new → Package → enter name = com.satya.catalog → finish.

Step3 → Right click on Package Name → new → Interface → enter name = Catalog → finish

```
Package com.satya.catalog;

@WebService(name = "Catalog", ~~portName = "CatalogPort"~~,
            targetNamespace = "http://www.satya.com")

Public interface Catalog
{
   @WebMethod(operationName = "catalogItems")
     Public List getCatalogItems (@WebParam(name = "catalogId") int
                                                        catalogId);
}
```

Step 4
→ Right click on package name → new → class → enter name = Item → finish

```
Public class Item
{
   Private int itemId;
   Private String itemName;
   Private int price;
   Prrate int catalogId;

   // setters & getters
}
```

Steps
→ Right click on Package Name → new → class → enter name = CatalogService.

```
@WebService(serviceName = "CatalogService", portName = "CatalogPort",
            endpointInterface = "com.satya.catalog.Catalog")
   Public class CatalogService implements Catalog

{
   @Override
   Public List getCatalogItems (int catalogId)
   {
```

```
List list = new ArrayList();

try
 {
    Class.forName
    Connection Con
    PreparedStatement pstmt = con.prepareStatement ("select * from
                                                    catalog where catalogId = ?");
    pstmt.setInt (1, catalogId);
    ResultSet rs = pstmt.executeQuery();
    While (rs.next())
      {
         Item i = new Item();
          i.setItemId (rs.getInt(1));
          i.setItemName (rs.getString(2));
          i.setPrice (rs.getInt(3));
          i.setCatalogId (rs.getInt(4));
          list.add(i);
      }
    rs.close();
    pstmt.close();
    con.close();
 }
  Catch (Exception e)
   {
      sopln (e);
    }
 return list;
}
```

Step 6 → Copy Jaxws-RI Jaxws-rt jars & ojdbc14.jar to the lib folder.

Step 7 → copy web.xml and sun-jaxws.xml to WEB-INF directory

Open sun-jaxws.xml and change implementation & url patterns attributes

implementation = "com.sathya.catalog.CatalogService"
url-pattern = "/catalogs" />

Step 8 → Generate the binding classes of webservice from command prompt using wsgen tool like the following

D:\W1\CatalogApp\build\classes > wsgen -cp . com.sathya.catalog.CatalogService

↑ Workspace name

→ Right click on Project name in eclipse ide → click on Refresh. Now binding classes are added to Project

→ Deploy this appln to the tomcat Server

Step 9

→ to see the wsdl file of the webservice, type the following url in address bar.

http://localhost:8888/CatalogApp/catalogs?wsdl

Step 10

→ Create a catalog table and insert some rows like the following

create table catalog (itemid number(5), itemname varchar2(10),
Price number(5), catalogid number(5));

21-5-2015

Creating a Console Appl^{n} as Client for Eclipse IDE

~~Client~~

Step 1 File menu → new → Java Project → next → Enter Project Name = CatalogClientProject → finish

Step 2 Right click on the Project Name → new → other → expand webServices in wizards → select a "webservice client" → next → Enter Service definition = "http://localhost:8888/CatalogApp/catalogs?wsdl → finish.

Step 3 Right click on src folder → new → package → enter name = com.satya.client → finish

Step 4 Right click on com.satya.client → new → class → Enter name = Main → finish

axis.apache.org → Apache Axis2 | Java → Downloads → Releases

axis.apache.org | axis2 | java | core | download.cgi

Apache AXIS-2

AXIS-2

→ AXIS-2 is a web services engine, it contains implementation of JAX-WS API

→ if we want to do use AXIS-2 implementation of Apache for creating a webservice and for deploying it on a server, we need to download binary distribution & war distribution of axis2 from

axis.apache.org | axis2 | java | core | download.cgi

→ before we deploy web service that uses axis-2 implementation, first we need to add axis-2 engine to the server

→ Extract ~~AXIS-2-1.6.2~~ ~~AXIS2-1.6.2 ZIP~~, AXIS2-1.6.2 war.zip

→ axis2.war will be extracted from the zip file.

→ deploy (copy) axis2.war to tomcat 7.0/webapps folder. now axis2 engine is added to tomcat

→ to know whether Axis2 engine successfully deployed in server or not, start the server & type the following request from browser

http://localhost:2015/axis2

22-5-15

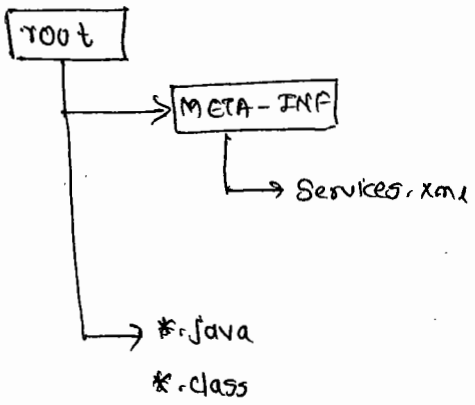
Creating a Webservice in axis2

→ for each webservice we need to create a Axis archieve file (aar file)

→ Axis Archieve file directory structure contains a root directory with a Sub-directory META-INF

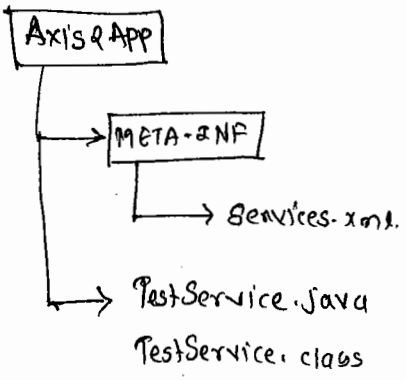
→ In META-INF directory, we store services.xml this file is used by Axis2 engine for generating binding classes and Wsdl file

→ the Source code and the class files are stored in root directory



Step 1

→ create a Directory structure for the Application



Step 2

→ create a TestService.java

// TestService.java

```
Public class TestService
{
   Public String sayHello (String str)
      {
        return "Hello: " +str;
      }
   Public String sayBye (String str)
      {
        return "Bye :" +str;
      }
}
```

Step3

→ Create Services.xml like the following

```
<!-- services.xml
<service name = "service1">
   <Parameter
      name = "ServiceClass"> TestService </Parameter>
      <operation name = "sayHello">
        <messageReceiver
          class = "org.apache.axis2.rpc.receivers.RPCMessageReceiver" />
      </operation>
      <operation name = "sayBye">
        <messageReceiver
          Class = "org.apache.axis2.rpc.receivers.RPCMessageReceiver" />
      </operation>
</service>
```

→ a MessageReceiver class is used by Axis2 webservices engine for generating binding classes that are only for converting a SOAP Request into Java method call or for both converting SOAP

Request to method call & converting a return value into SOAP Response.

→ Axis2 has provided 2 message Receiver classes RPCMessageReceiver and RPCInOnlyMessageReceiver.

→ If a WebService method has a return value then we need to configure RPCMessageReceiver

→ if a WebService method doesn't contain any return value then we need to configure RPCInOnlyMessageReceiver.

D:\ AxisApp

Axis2App > javac TestService.java.

Axis2App > jar cvf service1.aar

step 4
→ copy service1.aar file into

C:\Program Files\ Apache Software Foundation\ Tomcat 7.0\ WebApps\ axis2\ WEB-INF\ Services folder.

Note
↳ suppose, if any jar files are needed to run the service then we need to copy the jar into

Tomcat 7.0 \ Webapps \ axis2 \ WEB-INF \ lib folder

step 5
→ start the tomcat server and open the axis2 home page

localhost: 8888 / axis2 /

→ click on services link ⟶ click on service1
→ WSDL file generated by wsdl will be displayed.

Creating a Client Appl^n Using wsdl cosimport tool

Step 1

→ Create a folder Axis2Client

→ Generate client side Binding classes using cosimport tool like the following

D:\Axis2Client> cosimport -p pack1 -keep http://localhost:2015/axis2/services/Service1 ? wsdl

Parsing wsdl . . .

Generating code . . .

Compiling Code . . .

Step 2

→ Creating a client Appl^n like the following

```
import pack1.*;
class Client
  {
    main
    {
      Service1 service = new Service1();
      Service1PortType spt = service.getService1HttpSoap11Endpoint();
      String s1 = spt.sayHello("abc");
      String s2 = spt.sayBye("abc");
      SOP(s1);
      SOP(s2);
    }
}
```

D:\Axis2Client> javac Client.java

D:\Axis2Client> java Client

Hello : abc

Bye : abc.

25-5-2015

Creating a Web-Service with Axis-2 Implementation using Eclipse.

→ In eclipse IDE there is a Predefined plugin for adding Axis-2 runtime to the eclipse IDE Next

→ If Axis-2 runtime is loaded then eclipse makes our web Appln as a Access to container & deploys it in the server.

→ We can create a web services & web service client appln using Axis-2 Implementation with Eclipse easily

Step

① Start eclipse & enter some workspace name.

② click on window menu → preferences → expand Web Services → select Axis2 Preferences → Browse → select axis2 directory → ok

③ Click on → file menu → new → dynamic Web Project → Enter Project Name = DemoApp → click on new Runtime → Apache Tomcat v 7.0 → Select dynamic web module 2.5 → click on modify button & select Axis2 web Services. → finish.

④ Right Click on src directory → new → Package name = Com.sathya.axis2 → Finish

⑤ Right click on the Package name → new → enter name → DemoService

```
Public class DemoService
{
    Public String sayHello(String uname)
    {
        return "Hello --> "+uname;
    }
}
```

⑥ Right click on DemoApp → new → other → expand web Services → select web server → next → service type Bottom up → service implementation → com.sanya.axis2.DemoService →. click on web service runtime: link → select Apache Axis2 → or → next → select default Services.xml → next → start server → finish

⑦ to see the wsdl file type the following request in address bar

~~http://localhost~~:

http://localhost:2015/DemoApp/Services/DemoService?wsdl

Creating a client Application using Axis2 with eclipse IDE

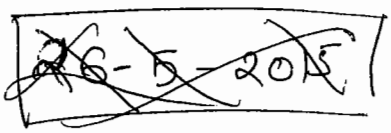
① → click on file menu → new → other → expand Web services → select web service client → enter service definition = http://localhost:2015/DemoApp/Services/DemoService?wsdl → click web service runtime: Apache Axis2 → select Apache Axis2 → click on Client Project → enter project name = ClientApp → ~~finish~~ Next → enter package name = Pack1 → finish.

② expand src Right click on Pack1 → new → class Name = Client → finish.

```
Package Pack1;
Public class client
{
    public static void main (String args[])
    {
        DemoServiceStub stub = new DemoServiceStub();
        DemoServiceStub.SayHello in = new DemoServiceStub.SayHello();
        in.setUname("xyz");
```

```
DemoServiceStub.SayHelloResponse out = stub.SayHello(in);
String str = out.get_return();
sopln(str);
}
}
```

→ Right click in Client class → run as → Select Java Appl?



JAX-B API

Version 2.x

26-5-2015

Java Architecture for Xml Binding

JAXP API
1. SAX API → only for reading
2. DOM API → reading & writing

Marshaller :- Java object → xml.
Unmarshaller : xml file → Java object
} JAXB

JAXB annotations.

~~Public class~~

→ JAXB API is an advanced API for JAXP API.

→ for reading an xml file content into a Java object & for writing a Java object's data into xml file, JAXP API has internaly 2 API's
1. SAX API
2. DOM API

→ Both SAX & Dom API's are heavy & they are complicated so to reduce the burdon, JAXB API introduced for converting Java object to xml & in reverse

→ JAXB API mainly contains 2 objects.
1. Marshallar
2. Unmarshallar

→ a Marshallar or marshallar object Java object into the xml file, and an Unmarshallar converts xml file into a Java object.

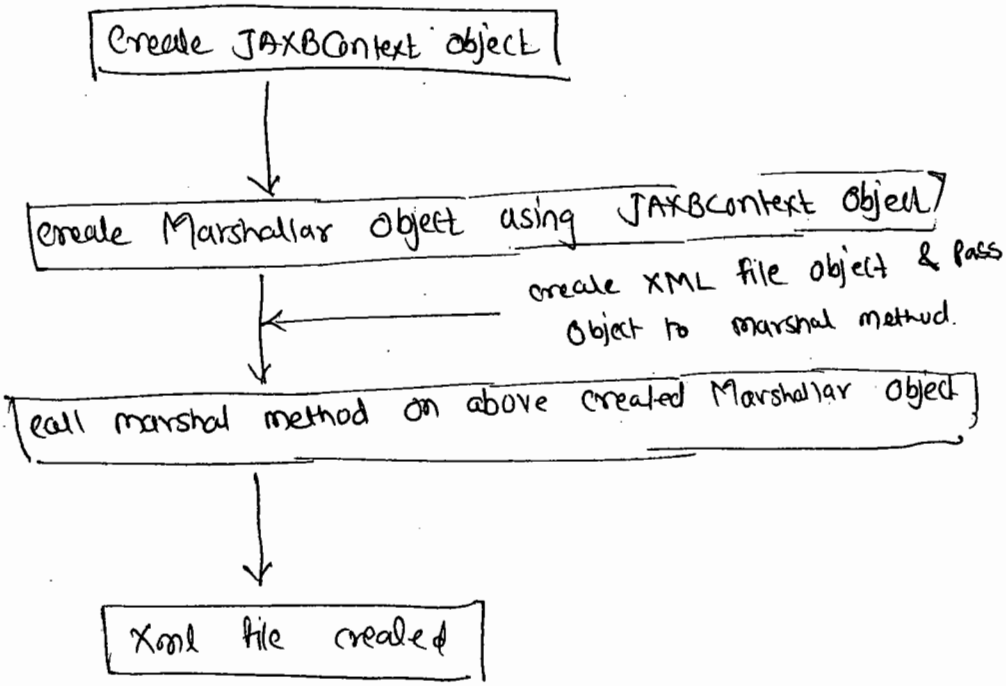
marshallar object

```
File file = new File ("c:/student.xml");
m.marshal (s1, file);
```

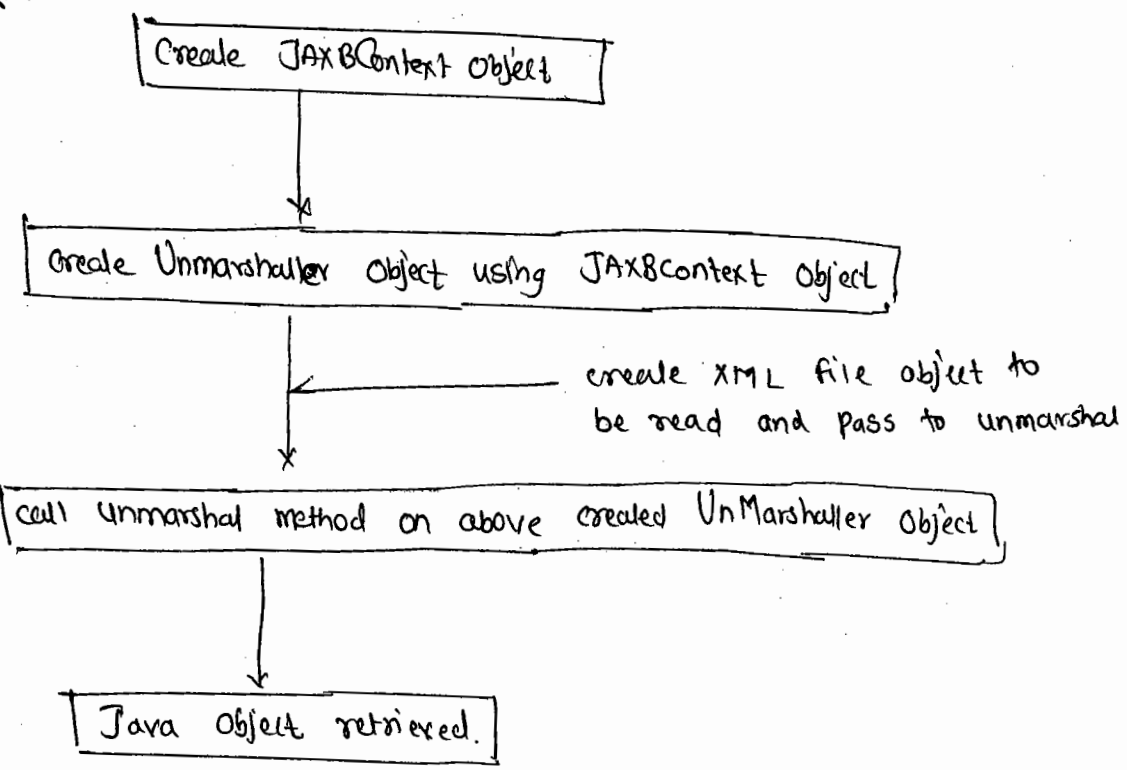
→ In JAXB API, a root object to enter into JAXB is JAXBContext object.

→ Using ~~JAXBCo~~ JAXBContext object only we can create a marshalar or unmarshallar object.

Marshallar Diagram



Unmarshaller Diagram



→ while converting a Java object into an xml file to control the elements that are going to be created in xml file, In JAXB API annotations are given

→ In the following example we want to convert 2 objects of students into a xml file

→ if we want to Convert multiple Object's of class into a xml file, we need to create another class for adding multiple object to a collection

→ In the following example, we creating a Students class with a Collection variable for adding 2 objects of student class

JaxbExample1

- Student.java
- Students.java
- JavaToXml.java

```
//Student.java                    // Student.java
  Package com.sathya.jaxb;
 import javax.xml.bind.annotation.XmlType;
@XmlType (propOrder = {"studentId", studentName", "marks"})

 Public class Student
   {
      Private int marks;
      Private int studentId;
      Prirate String studentName;

     // setters & getters
   }
```

```
// Students.java
 Packag com.sathyer.jaxb;
import —
import —
 @XmlRootElement
  Public class Students
  {
    Private List<Student> studentList;
    Public void setStudentList (List<Student> studentList)
      {
        this.studentList = studentList;
      }
```

```
@XmlElement (name = "student")
  Public List <Student>  getStudentList ()
    {
      return studentList ;
    }
};
```

// JavaToXml.java

```
Import Javax.xml.bind.*
import com.sathya.jaxb.Student ;
import com.sathya.jaxb.Students ;
      Java.io.file ;
      Java.util.List ;
      Java.util.ArrayList;

Public class JavaToXml
  {
    main () throws Exception                    Static factory method
                                                      ↓
  {
    JAXBContext jaxbContext = JAXBContext.newInstance (
                                          Students.class);

      Marshaller marshaller = jaxbContext.createMarshaller ();

    Student student1 = new Student ();
    student1.setStudentId (101);
    student1.SetStudentName ("ABC");
    Student1.setMarks (500);

    Student student2 = new Student ();
```

} repeat it for 1 time.

```
List <Student> studentList = new ArrayList<Student> ();

    studentList.add (student1);
    studentList.add (student2);

    Students students = new Students ();
    students.setStudentList (studentList);

    File file = new File ("E:/student.xml");
    marshaller.setProperty (Marshaller.JAXB_FORMATTED_OUTPUT, true);

    marshaller.marshal (students, file);
  }
}
```

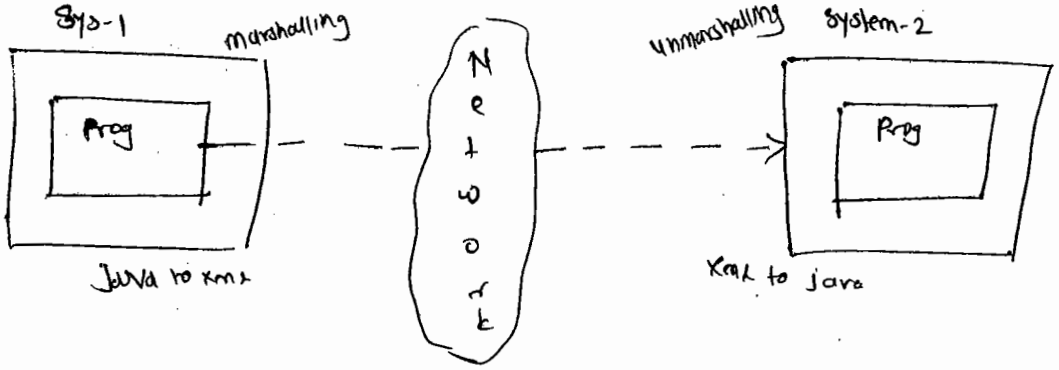
```
JaxbExample1 > javac -d . Student.java
             > javac -d . Students.java
             > javac JavaToXml.java
             > java JavaToXml
```

29-5-2015

→ the following Program converts xml file into Java object using Unmarshaller of JAXB

```
//XmlToJava.java

import javax.xml.bind.*;
import com.satnya.jaxb.Student;
import com.satnya.jaxb.Students;
        java.io.File;
        java.util.List
        java.util.Iterator;

Public class XmlToJava
  {
     main() throws Exception
                                          class object of Class
                                                ↓
   {
     JAXBContext jaxbContext = JAXBContext.newInstance(Students.class);
     Unmarshaller unmarshaller = jaxbContext.createUnmarshaller();
     File file = new File("E:|student.xml");
     Students students = (Students) Unmarshaller.unmarshal(file);
     List<Stud
     List<Student> list = students.getStudentList();
     Iterator it = list.iterator();
     while (it.hasNext())
        {
          Student student = (Student)it.next();
          SOPln("id :" + student.getStudentId());
          SOPln("name:" + student.getStudentName());
          SOPln("marks:" + student.getMarks());
```

```
        System.out.println("======");
    }
}
```



Importance of XSD file in JAXB

XSD to Java Compiler (xjc)

* Importance of XSD file in JAXB

→ Using JAXB API if we are marshalling and unmarshalling in the same system then there is no need of sending xml file and the .class file In a network.

→ Suppose we want to marshal in one system & we want to unmarshal in another system of network then we need to transfer not only xml file but also java classes in network.

→ transferring a large number of java classes in network is not good so a solution is found in the form of XSD file.

→ Instead of transfering Java classes, we can transfer only XSD file in network and Based on XSD file java classes can be created.

→ the importance of xsd file in JAXB is, at both marshalling and unmarshalling side same java classes will be used. there is no chance of getting exception in unmarshalling.

→ In the following Example we are creating employees.xsd file & we are generating java classes based on xsd file, to marshal the java objects.

JaxbExample2

employees.xsd

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xs:element name="employees" type="employees" />
  <xs:complexType name="employees">
    <xs:sequence>
      <xs:element name="employee" type="employee"
                  minOccurs="1" maxOccurs="3" />
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="employee">
    <xs:sequence>
```

```
<xs: element name = "name" type = "xs: string"
            minOccurs = "1" maxOccurs = "1" />

<xs: element name = "designation" type = "xs: string
            minOccurs = "1" maxOccurs = "1" />

<xs: element name = "address" type = "address" />

</xs: sequence>

<xs: attribute name = "id" type = "xs: int"
            use = "required" />

</xs: complexType>

<xs: complexType name = "address">
  <xs: sequence>
    <xs: element name = "street" type = "xs: string" />
    <xs: element name = "city" type = "xs: string" />
    <xs: element name = "state" type = "xs: string" minOccurs = "0"

  </xs: sequence>
</xs: complexType>
</xs: schema>
```

JaxbExample2 > xjc -p com.pack1 employees.xsd

Parsing a schema

Compiling a schema

com\pack1\Address.java
com\pack1\Employee.java
com\pack1\Employees.java
com\pack1\ObjectFactory.java

JaxbExample2\com\pack1 > javac *.java

Note: Open Employee.java and add @XmlRootElement on top of the class and also add its import statement on top of the class.

Marshal.java

```
import javax.xml.bind.*;
import com.pack1.*;
class Marshal
    {
    public static void main() throws Exception
    {
    JAXBContext jaxbContext = JAXBContext.newInstance("com.pack1");
    Marshaller m = jaxbContext.createMarshaller();

    Address address = new Address();
    address.setStreet("Ameerpet");
    address.setCity("Hyd");
    address.setState("TS");

    Employee e = new Employee();
    e.setId(101);
    e.setName("ABCD");
    e.setDesignation("Manager");
    e.setAddress(address);

    Employees es = new Employees();
    es.getEmployee().add(e);
```

```
m.setProperty(Marshaller.JAXB_FORMATTED_OUTPUT, true);
m.marshal(es, new Java.io.File("f:/emp.xml"));
}
}
```

~~JAXB~~

JaxbExample2 > Javac Marshal.Java

> Java Marshal

28-5-2015

SOAP UI

→ SOAP UI is a GUI tool used by a webservice provider or webservice client for testing operations of a webservice.

→ a Service provider will test wheather operations are working properly or not before releasing url of a ~~dd~~ wsdl to the clients

→ a client side developers uses SOAP UI tool for testing wheather operations are working properly or not before creating & or constructing client Application.

→ SOAP UI is an open source & SOAP UI PRO is commercial

We can download SOAP UI from www.sourceforge.net/projects/SOAPUI

an exe file is downloaded & we need to installed the application

webservicex.net/ws/default.aspx

→ the following steps are for testing operation of a webservice currency converter created by 3rd party

webservicex.net

→ we can find the list of webservices given by 3rd party webservicex.net. we need visit www.webservicex.net.

step

① launch SOAP UI window

② Right click on projects, select a new SOAP Project → enter Project name (Project1) → OK

③ Right click on Project1 → Add wsdl → enter wsdl location = http://www.webservicex.net/currencyConverter.asmx?WSDL

④ expand conversionRate (operation name) → double click on request 1

→ enter FromCurrency ⇒ USD
ToCurrency ⇒ INR

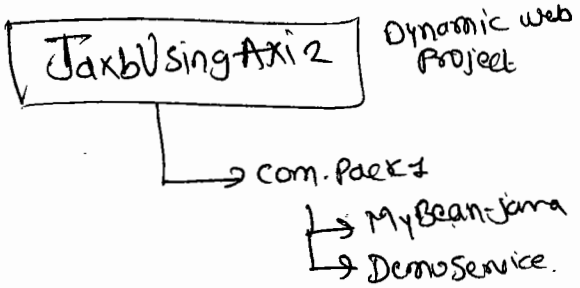
→ click on submit Request button, & Right side we will get Response.

```
P  C  DemoService
{
    public MyBean getHike(MyBean mb)
    {
        'ip is MyBean object & op is also MyBean Object
    }
}
```

p c MyBean implements Serializable

{
name
value
setter/getter
}

⇒ In the following, we are creating a webservice using Axis-2 implementation which takes Input as a java class object and returns output as a same object. using eclipse.



MyBean.java

```
Package com.pack1;

import java.io.serializable;

Public class MyBean implement Serializable
{
    Private String name
    private int value;

    // setters & getters

}
```

DemoService.java

```
Package com.pack1;

Public class DemoService
{
    public MyBean getlike (MyBean mb)
    {
        int i = mb.getValue();
        i++;
        mb.setValue(i);
```

return mb;

}

}

29-5-2015

first wsdl & based on this generate java classes

Contract first service last
i.e Top down Approach
(rarely used)

Service first contract last
i.e ~~Bot~~ Bottom up Approach

Example → Eclipse name ⇒ Test_WebService_TopDown Project Name

→ Creating a WebService Top down (Contract first Approach)

- Add runtime → Axis-2 Preferences
- Add Tomcat Server if not added

~~java Project → only create~~

Select Dynamic web Proj

- new → expand web service → wsdl file → enter file name

if Demo.wsdl

enter Target Namespace →

→ select create wsdl skeleton

→ document literal

* * *

add tool SoapUI in Resume ← Testing purpose wsdl file.

Step's

① Add Axis-2 Preferences to eclipse IDE

Click on file menu → new → select dynamik web ~~Proj~~ Project

→ enter project Name = Test → module version 2.~~J~~ finish

② Right click on the project Name → New → other → expand webServices

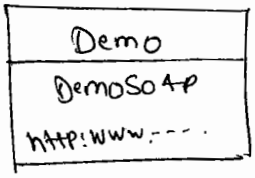
→ select wsdl file → enter file name → Demo.wsdl →

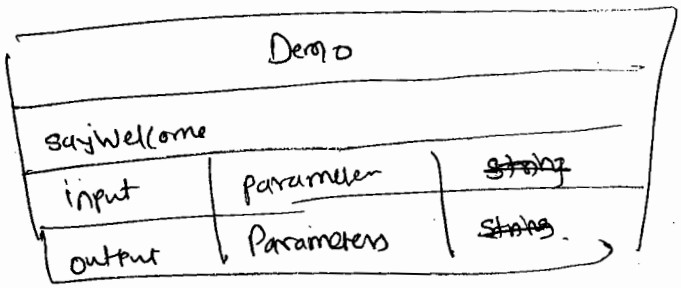
targetNamespace = http://www.sathya.com/demo → Prefix → ths

→ select WSDL ~~Sk~~ Skeleton → select document ~~Li~~ literal finish

③ Demo.wsdl design is opened.. Double click on New Operation

word and change it to ~~Say~~ sayWelcome. Right click on ~~input~~





~~Say~~ sayWelcome input SetType → existing type → select string → ok

Similarly sayWelcomeResponse SetType → existing type → select String → ok

Note

→ we can see the wsdl xml by clicking on source ~~tag~~ tab

④ → Right click on project name → new → webservices → webService →

Service type = Top Down → Browse → Browse button → ~~Server~~ select

Demo.wsdl. → click on web Service runtime → Select Apache ~~Axis~~ 2 → OK

→ Next → enter custom package name = com.sathya.pack1

→ Start Server → finish.

Step 8 DemoSkeleton.java is created add the following code in sayWelcome() method of the skeleton class

```
Package com.satya.pack1;

Public class DemoSkeleton
 {
   Public Com.satya ---- ——
     {
      String str1 = sayWelcome.getIn();
      String str2 = str1.concat ("satya");
      SayWelcomeResponse response = new SayWelcomeResponse();
                                    response.setOut (str2);

      return response;
      }
 }
```

Step 9 Right click on project name run as → run on server → finish

20-5-2015

Q. What is SOAP fault?

→ SOAP fault message (server side) —converted to→ client language Exception (client side)

SOAP Exception Handling

→ When a client calls operation of a webservice, if an Exception is thrown or occured in operation of webservice then directly that Exception will not be thrown directly back to the client application. because client and server Appl'n may not be in same language.

→ if an Exception occurred in a webservice then binding classes at server side will convert that Exception into a SOAP fault and then fault message inserted in SOAP Body response & finally respone will sent back to the client Application.

→ at a client side the binding classes converts a SOAP fault into a Exception of that client side language and it will be thrown to the client Application.

→ A SOAP fault message contains 4 Parts

1. fault code.
2. fault string
3. fault actor
4. detail

→ Example

Soap response

```
<soap: Envelope>
    <soap: Body>
        <soap: fault>
            <faultCode> xxx </faultCode>
            <faultString> xxx </faultString>
            <faultactor> xxx </faultactor>
            <detail> xxx </detail>
        </soap: fault>
    </soap: Body>
</soap: Envelope>
```

Important

→ the body part of soap response can have either return value of webservice operation or a soap fault message.

→ if webservice operation is successfully executed then body of soap response contains return value. other wise it contains a soap fault message.

→ a fault code can be soap: client / soap: server: / soap: versionmismatch.

→ if there is a invalid input value sent by the client then fault code is soap: client. if exception is occurred in webservice the fault code is soap: server. if soap version used by client is 1.1 and the server version is 1.2 then the fault code is a soap: versionmismatch.

→ a fault string is a Human readable message of the exception.

→ a fault actor is the url of the web service.

→ detail section contains exception object thrown by the web service in the form of xml

→ according to JAXWS specification, if we want to throw an Exception then we need to create a Exception class & also a fault bean class (it is just a normal bean class).

→ In Exception class, we need to define two constructors with two arguments, & 3 arguments and then a method getFaultInfo()

→ on Top of the Exception class we must add @WebFault annotation.

** for Example

→ In the following webservice, we are throwing a missing name Exception if input value is not set by the client

```
@WebService
Public class MyWebService
{
    @WebMethod
    Public String sayHello (String str) throws MissingNameException
    {
        if (str.length() == 0)
            throw new MissingNameException ("input value is
                must", new MissingBean());
```

```
        else
            return "Hello" +str;
    }
}
```

this is given according to the JAX-WS specification.

```
@WebFault
Public class MissingNameException extends RuntimeException
{
    Private MissingBean mb;
    Public MissingNameException(String message, MissingBean mb)
    {
        Super (message);
        this.mb = mb;
    }
Public MissingNameException (String message, MissingBean mb,
                             Throwable t)
    {
        Super (message);
        this.mb = mb;
    }
Public MissingBean getFaultInfo()
    {
        return mb;
    }
}
```

```
Public class MissingBean  → this called fault bean
{
    Private String message;
    Public void setMessage (String message)
    {
        this. message;
    }
    Public String getMessage()
    {
        return message;
    }
}
```

1-6-2015

Creating asynchronous client.

→ In Soap web services we have two types of clients

1. Synchronous client
2. asynchronous client.

→ Synchronous clients means. a client Appl^n should wait for the response of a webservice.

→ asynchronous clients means a client can continue its execution, by without waiting for the response. of a web service.

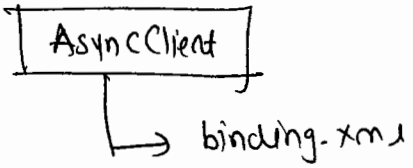
→ to create asynchronous client Appl^n, we need to prepared a Binding xml file & we need to pass this xml as a Parameter to wsImport tool.

→ wsimport tool generates binding classes for both synchronous and asynchronous for the webservice

→ In the following we are creating 2 main classes, one for Synchronous client and other for Asynchronous client to the JAX-WS-RI webservice we previously created.

Step 1

create binding.xml and save it in a AsyncClient folder




```
<bindings
    xmlns:xsd = "http://www.w3.org/2001/XMLSchema"
    xmlns:wsdl = "http://schemas.xmlsoap.org/wsdl/"
    wsdlLocation = "http://localhost:2015/BookTest/books?wsdl"
    xmlns = "http://java.sun.com/xml/ns/jaxws">
    <bindings node = "wsdl:definitions">
        <package name = "packt" />
        <enableAsyncMapping> true </enableAsyncMapping>
    </bindings>
</bindings>
```

Step 2 Generate the client side binding class. using wsimport tool like the following

```
D:\AsyncClient> wsimport -p packt -keep -b bindings.xml
    http://localhost:2015/bookTest/books?wsdl
```

→ after generating client side binding classes we can open Book.java (i), to find bookPriceAsync() method, this method is to call a webservice asynchronously.

→ async() method returns javax.xml.ws.Response object

→ Response object contains a method isDone() & it returns true when response is ready from webservice otherwise returns false

```
import Packt.*;
import javax.xml.ws.Response;
Public class Main2
  {
      main() throws Exception
    {
      BookService service = new BookService();
      Book b = service.getBookPort();
      Response res = b.bookPriceAsync("IS101");
      while(res.isDone() == false)
        {
          SopIn("Hello");
          try
           {
             Thread.sleep(10000);
           }
          Catch(Exception e)
           {
           }
    }
      BookPriceResponse r = (BookPriceResponse) res.get();
      double d = r.getReturn();
```

```
SOPln ("price = " +d);
}
```

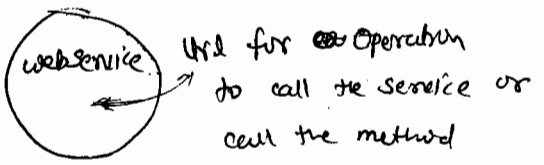
Q-5- 2015

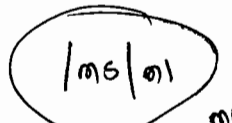
Rest

RESTful Webservices

↳ REPresentation state Transfer

→ REST is totally light weight






```
@Path ("ms")
Public class MyService
{
    @Path ("m1")
    Public String sayHello()
    {
        return "Hell";
    }
}
```

Servlet url Patten

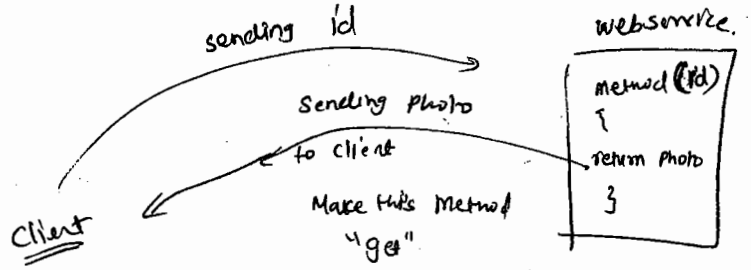
http://localhost:8898/App/rest/ms/m1

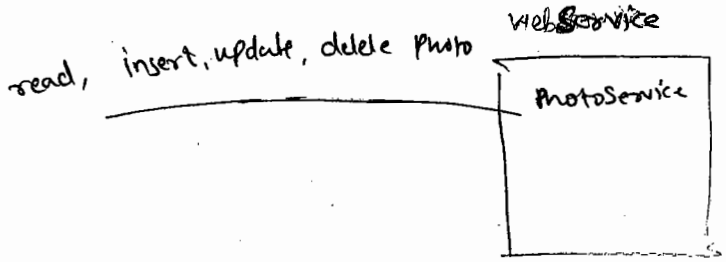
uri

url

- documentry is made for all url, I/P & O/P.

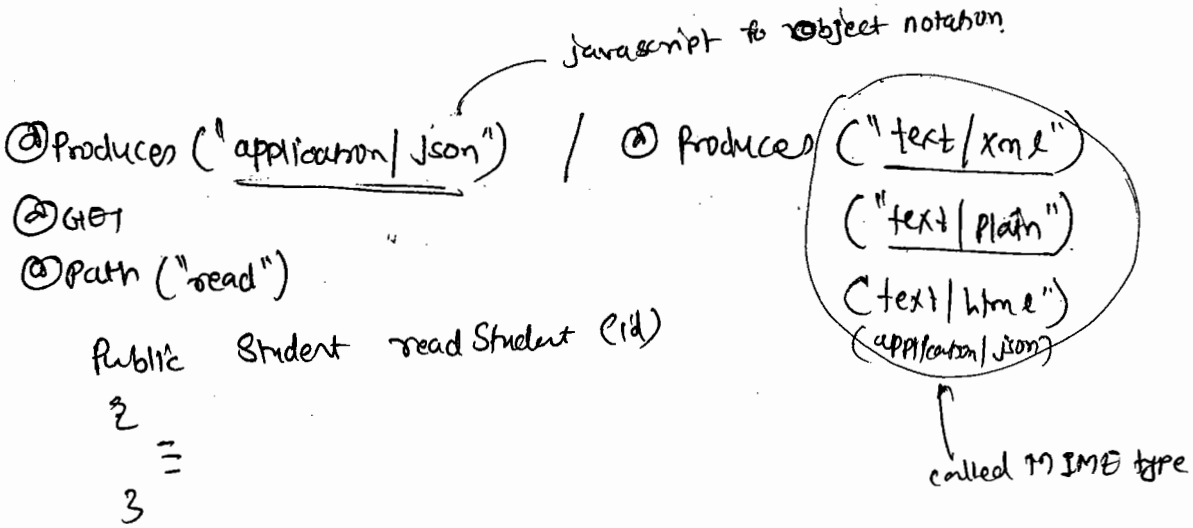
→ Provide it to the client






```
Public class PhotoService
{
    @GET
    @Path ("read")
    Public photo readPost (String id)
    {

    }

    @Post
    @Path ("update")
      public string updatePhoto (id, Photo)
    {

    }
  @Put
  @Path ("insert")
  Public boolean insertPhoto (photo)
  {
  }=
```

```
@DELETE
@Path ("delete")
Public boolean deletePhoto (String id)
{
  ==
 }
}
```



→ Representation (State) Transfer.
data

→ SOAP Based webServices are Heavy weight, because of multiple xml files, binding classes & wsdl file

→ to develop a webservice easily i.e to make a webservice as a light weight another architecural style was design with name RESTful

→ a RESTful webService makes a use of HTTP protocol only to allow a client Application to access methods of RESTful webservices

→ In a RESTful webservices, a webservice class is called ~~root~~ rootresource class & each method of the class is called a resource.

→ to make a class as a RESTful webservice, it should follow the below principle

① each resource (method) of a webservice should contain a reachable uri (addresable uri)

② we need to define resources in a webservice class like the following

(i) if a resource (method) is created to return some data from server side to the client then make that resource as accessible to "GET" method of Http.

(ii) if a resource is created to modify the data at a server side then make it as accessible through "POST" method of Http.

(iii) if a resource is created to add the data to the server side then make it as accessible to "PUT" method of Http.

(iv) if a resource is created to delete the data from server side then make it as accessible to "DELETE" method of Http.

③ a resource of webservice class should use one of the following types as its MIME type.

(i) text/html.
(ii) text/xml
(iii) application/json
(ix) text/plain.

④ a Webservice class must be Public and a It must Contain default constructor.

⑤ a webservice class must be within a Package.

3-6-2015

Diff. bet^n SOAP & RESTful Webservice

Simple object Access Protocol
means (State transfer)

SOAP

given by WS-I

RE presentational (State) Transfer

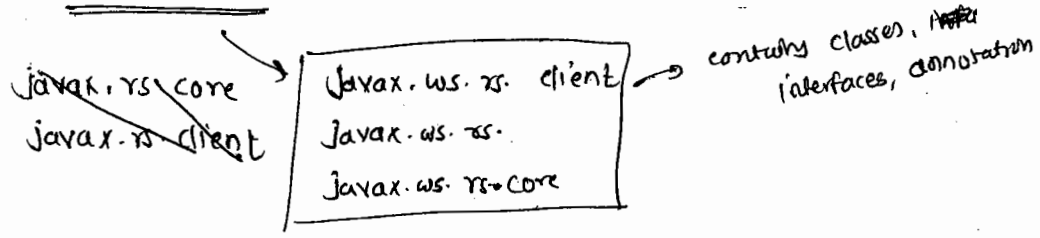
data

RESTful

directly API is given SUN

SOAP	RESTful
① SOAP has a specification	① no specification
② SOAP is a heavy weight Webservice, because of wsdl file and binding classes.	② REST is a light weight, because there is no need of binding classes & wsdl file.
③ If any class is modified or added in webservice then again make wsdl file & generate binding classes, it is not easy to migrate existing class as a SOAP webservice	③ It is easy to migrate an existing class as RESTful webservice.
④ In SOAP, a client Application can receive only data, but not its representation.	④ In RESTful, a client Application can receive data with representation
⑤ In SOAP, if we want to test a Webservice, we need to write a client Program or we need to install SOAP UI tool	⑤ In RESTful, we can directly send request from browser, to test a Restful web service.

JAX-RS - API



- Jersey ← sun
- RESTEasy ← jboss
- Restlet
- Triaxrs

} vendor

→ for creating a Restful webservice and also for creating client Applications sun microsystem released JAX-RS-API.

→ JAX-RS-API mainly contains 3 packages

1. javax.ws.rs.client → not available in jersey 1.x but available in jersey 2.x
2. javax.ws.rs
3. javax.ws.rs.core

→ the implementation of JAX-RS API is provided by vendors. some implementations are

1. Jersey ← given by sun
2. RESTEasy ← given by Jboss
3. Restlet
4. Triaxrs etc.

Annotations for creating a WebService

→ this Annotation is used for adding or attaching a uri for root resource class (webservice class) and also for resource (method)

① @Path

no compulsory

```
@Path ("/demo")
Public class DemoService
{
    @Path ("/hello")
    Public String sayHello ()
    {
        return "hello";
    }
}
```

→ Uri to call sayHello() is `/demo/hello`

→ Uri to call sayHello() method by a client Appl" is

`http://localhost:9898/App1/rest/demo/hello`

Uri Pattern of Servlet used to receive a request

→ In RESTful webservices, a request comes from client Application through Http Protocol. In order to receive a request i.e coming with Http Protocol, we configure a servlet given by the vendor in web.xml.

② @PathParam

→ this annotation is used for injecting a PathParameter value to a method parameter.

→ if we want to include any Path Parameters then we need to put Parameter Name in {}.

→ if Path Parameter name & method Parameter name is matched then passing Path Parameter name in a annotation is optional.

→ Example 1

```
@Path("/demo")
Public class DemoService
{
  @Path("/hello/{aname}")
  Public String sayHello (@PathParam("aname") String s1)
  {
    return "Hello" +s1;
  }
}
```

→ Url to call sayHello method from client appl" is

http://localhost:8888/App1/rest/demo/hello/sathya

Example: 2

```
@Path ("Sample")
Public class SampleService
{
```

```
@Path ("/name/{fname}-{lname}")
    Public String getName (@PathParam String fname,
                           @PathParam String lname)
    {
      String fullName = fname.concat (lname);
       return fullName;
    }
}
```

→ url to call getName from Client Application is

http://localhost:9898/App1/rest/sample/name/ Sathya - Java
                                              ↑ fname ↑ lname

③ @QueryParam

→ this annotation is used to inject a query parameter value to a method parameter

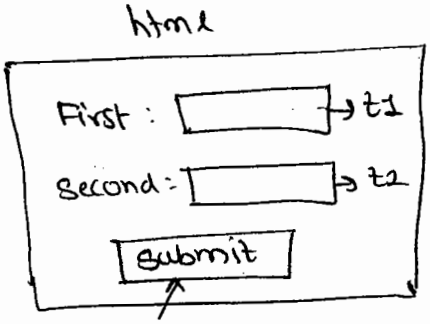
```
@Path ("Sample")
  Public class SampleService
  {
    @Path ("name")
    Public string getName (@QueryParam ("fname") String s1,
                           @QueryParam ("lname") String s2,
    {
       String s3 = s1.concat (s2);
       return s3;
    }
}
```

→ url for calling the getName() method from client Appln is

http://localhost:9898/APP1/rest/sample/name?fname=sathya&lname=Java

4. @FormParam:

→ this annotation is used for injecting the value of Html form Parameter to a method Parameter



→ for Example

```
@Path("sample")
Public class SampleService
{
 @Path("name")
 Public String getName(@FormParam("t1")String s1,
                       @FormParam("t2") String s2)
 {
    String s3 = s1.concat(s2);
    return s3;
 }
}
```

5. @Produces:

(servlet container)

→ this annotation is to tell a Restful container ^ about to which MIME type response should converted

~~When it~~ s

```
@Produces ("application/json")
@Path ("read/{id}")
Public Student readStudent (@PathParam String id)
{
    =
}
```

↓ produces service in given format i.e from web service to client

→ In the above RESTful container converts Student object to json object format and then that Response will be sent back to the client.

4-6-2015

6. @Consumes

→ this annotation tells (Application) about what type of input in what format accepted from the client by a webservice method.

→ when a client is calling a method then that client Appl^n must send a ip to the method in its acceptable format.

means in which format server receive the request from client.

→ @Path ("Insert")

@Consumes ("text/xml")

~~@GET~~ @PUT

Public String insertStudent (Student s)

{

=

}

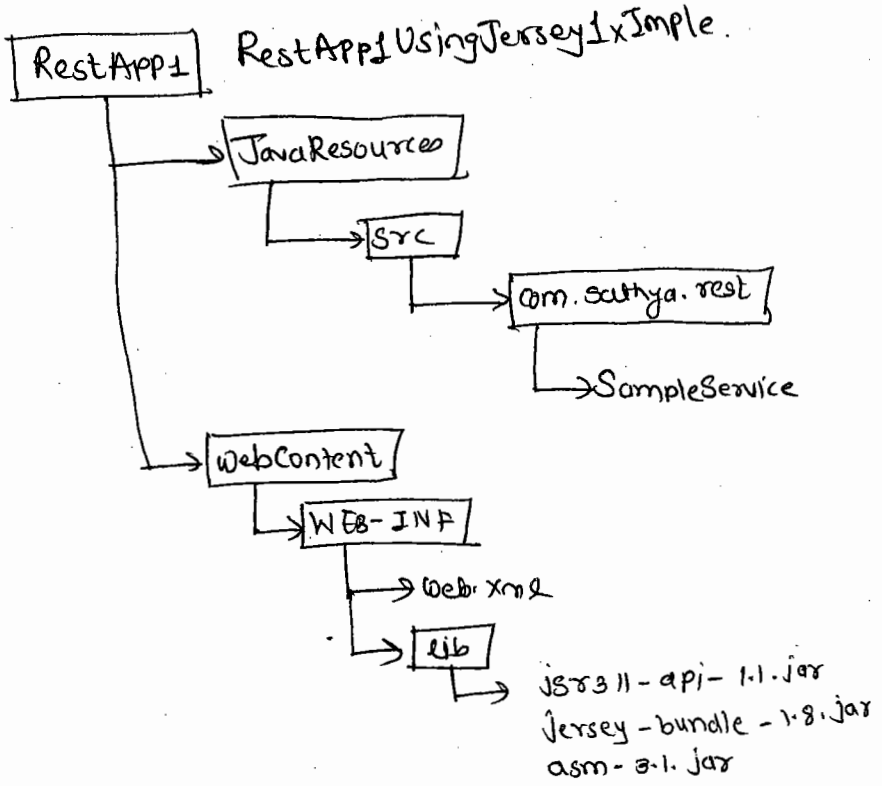
→ the annotation related to Http Protocol method are @POST, @PUT, ~~@UPDATE~~, @DELETE, @GET

jersey.java.net resteasy.jboss.org

javax

Example 1 Using jersey 1.x implementation

@ eclipse project structure



```
Package com.sathya.rest;
import javax.ws.rs.GET;
import javax.ws.rs.Path
import javax.ws.rs.PathParam;
import javax.ws.rs.Produces;
@Path("sample")
Public class SampleService
{
   @GET
  @Path("hello/{uname}")
  @Produces("text/html")
 Public String sayHello (@PathParam("uname") String uname)
  {
    return "<h1> Hello : "+uname+"</h1>";
  }
 @GET
@Path("bye")
@Produces("text/plain")
 Public string sayBye ()
    {
      return "<h1> Bye.....</h1>";
    }
}
```

Web.xml

```
<web-app>
<servlet>
    <servlet-name> jersey-servlet </servlet-name>
    <servlet-class> com.sun.jersey.spi.container.servlet.ServletContainer </s-c>
     <init-param>
       <param-name> com.sun.jersey.config.property.packages </param-name>
       <param-value> com.sathya.rest </p-v>
    </init-param>
   <l-o-s> 1 </l-o-s>
</servlet>
<servlet-mapping>
    <servlet-name> jersey-servlet </s-n>
    <url-pattern> /rest/* </u-p>
</servlet-mapping>
</web-app>
```

Request

http://localhost:9898/RestApp1/rest/sample/hello/ABCD → this url is for calling sayHello() method

http://localhost:9898/RestApp1/rest/sample/bye → url for calling sayBye() method

⇒ In ~~JAXRS~~. JAX-RS-1.X API there is no Separate API is given to develop a client Application in order to call a Restful webservices.

→ We should use java networking API (java.net package) for developing a client Application.

→ In JAXRS-2.X API client API is also given under the package javax.ws.rs.client. so we can develop client Appln in java easily.

converted into

Url Pattern "/rest/*"

④ ApplicationPath("rest")

```
Public class MyApplication extends Application
{

}
```

JAXRS-2.X API

5-6-2015

→ In JAX-RS-2.X api an alternat way is given instead of configuring a Vendor provided servlet class in the web.xml

→ Instead of writing web.xml, we need to create a class by extending from Superclass Application and we need to add @ApplicationPath annotation for the class.

→ In Application class we need to override getSingletons() method which returns a set with object of root resource classes.

```
@ApplicationPath("rest")        ← url pattern to Restful container
Public class MyApplication extends Application
{
  Private Set s;
  Public MyApplication()
  {
    s = new HashSet();
    s.add(new SampleService());
    s.add(new DemoService());
  }
  Public Set getSingletons(){
    return(s)        → return Set object which contain a list of service classes.
  }
```

this is Restful container

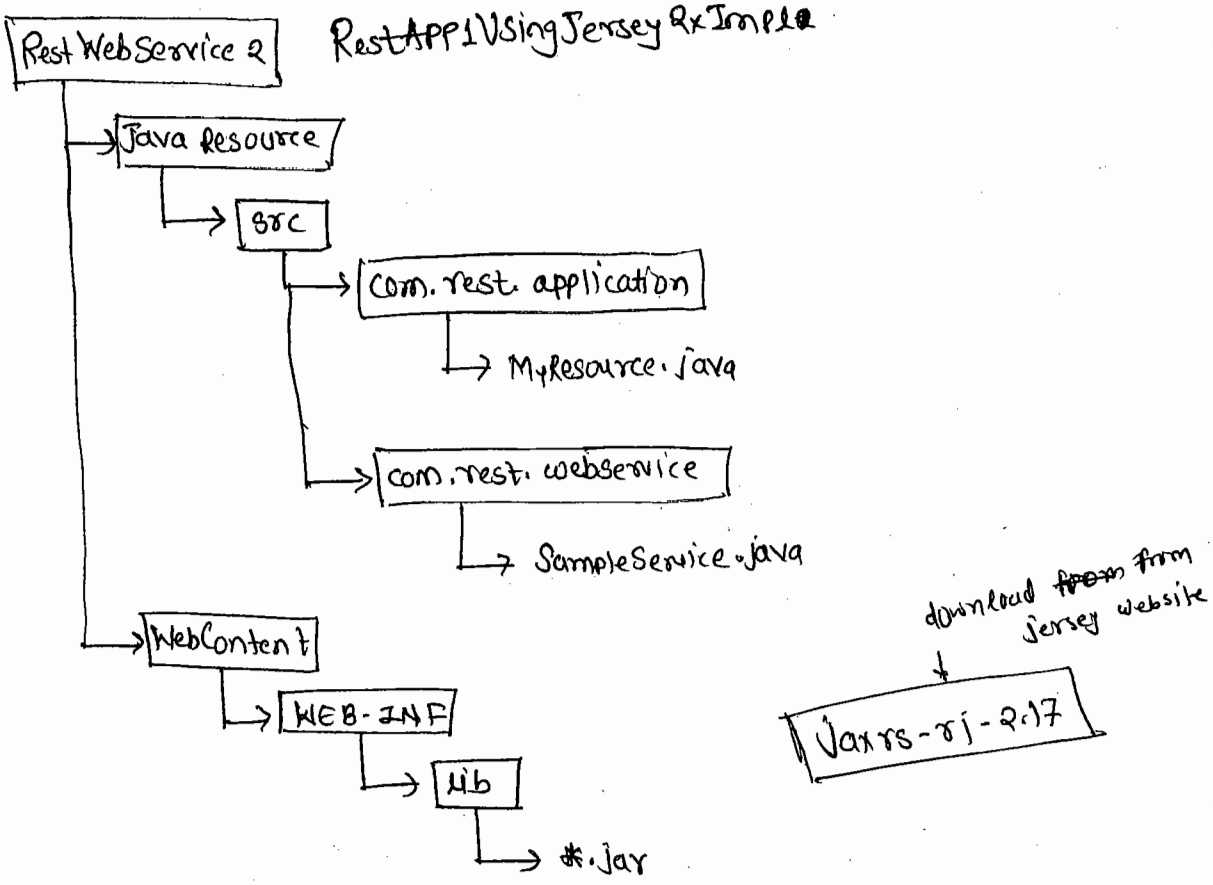
ServletContainer ← receive request

request

Server ← client

Example 2

→ In the following Example we are creating a Restful webservice using Jax-RS 2.x Api with Jersey implementation 2.x



→ first add following list of jar file to the lib folder

- ✓ asm-8.1.jar } separately download.
- ✓ cglib-2.2.2.jar }
- ✓ hk2-api-2.2.0 jar
- ✓ hk2-locator-2.2.0.jar
- ✓ hk2-utils-2.2.0.jar
- ✓ javaassist-3.18.1-GA.jar
- ✓ javax.annotation-api-1.2.jar
- ✓ javax.inject-2.2.0.jar
- ✓ javax.ws.rs-api_2.0.jar
- ✓ jersey-client.jar
- ✓ Jersey-common.jar
- ✓ Jersey-container-servlet-core.jar
- ✓ Jersey-container-servlet.jar
- ✓ Jersey-guava-2.8.jar
- Jersey-server.jar
- ✓ validation-api-1.1.0.final.jar

→ When we extract a zip file downloaded from Jersey-2.x version then we will get 3 subfolders ~~and the jar~~ under ~~jaxrs~~ Jaxrs-ri folder.

① api
② ext
③ lib

→ we can copy the jar files from subfolder

// MyResource.java ← it is alternate way to write web.xml

```
Package com.rest.application;
import java.util.HashSet;
@ApplicationPath ("rest")
Public class MyResource extends Application
{
    Private set s;
    Public MyResource ()
    {
        s = new HashSet();
        s.add (new SampleService ());
    }
    @Override
    Public Set getSingletons ()
    {
        return s;
    }
}
```

// SampleService.java

```
package com.rest.webservice;

import =

@Path ("sample")

Public class SampleService
  {
    @GET
      @Produces ("text/html")
      @Path ("{username}")
    Public String sayHello (@PathParam ("username") String s1)
    {
    return "<font color='blue' size=8 > Hello:" +s1 + "</font>";
    }

    @GET
      @Produces ("text/plain")
       Public String sayBye()
        {
         return "Bye";
        }
  };
```

String str = (String) target.request().get(String.class);

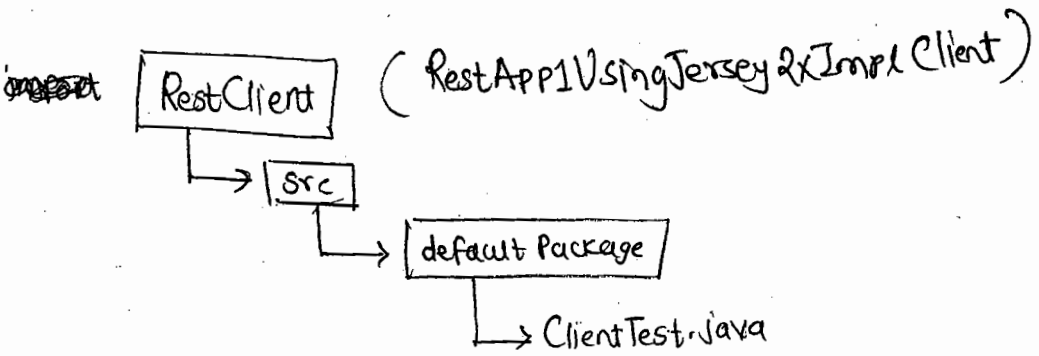
class object of String class

calling sayHello()

OR

```
Response obj = target.request().get();
String str = (String) obj.readEntity(String.class);
```

Creating a Java Client Application for the above RESTful Webservice using Jaxrs client API




```
// ClientTest.java
import javax.ws.rs.client.Client
import      -||-      .ClientBuilder
import      -||-      .WebTarget;
import javax.ws.rs.core.Response;

Public class ClientTest
  {
    main()
    {
      Client client = ClientBuilder.newClient();
      WebTarget target = client.target("http://localhost:8888/RestWebService2
                                        /rest/sample");

      target = target.Path("{username}").resolveTemplate("username",
                                                         "Sathya");
{ // String str = (String) target.request().get(String.class);
or
  {
      Response obj = target.request().get();
      String str = (String) obj.readEntity(String.class);
      SOPln(str);
      SOPln("=========");
```

WebTarget means url upto root resource class

```
WebTarget target2 = client.target("http://localhost:8888/RestWebService2
                                  /rest/sample");
String str2 = (String) target2.request().get(String.class);
System.out.println(str2);
}
}
```

→ Configure following jars to the build Path-

- aopalliance-repackaged-2.4.0-b10.jar
- hk2-api-2.4.0-b10.jar
- hk2-locator-2.4.0-b10.jar
- hk2-utils-2.4.0-b10.jar
- javax.annotation-api-1.2.jar
- javax.inject-2.4.0-b10.jar
- javax.ws.rs-api-2.0.1.jar
- jersey-client.jar
- jersey-common.jar
- jersey-container-servlet-core.jar
- jersey-container-servlet.jar
- jersey-guava-2.17.jar
- jersey-media-jaxb.jar
- jersey-server.jar
- validation-api-1.1.0.Final.jar

→ the entry point for creating a client Application using client Api of Jaxrs is Client object.

→ Client is an interface and we can get an object this type by calling static factory method newClient() of an abstract class ClientBuilder.

→ To call a resource of a root resource class, we need a WebTarget object for the url.

→ We can append a Path to a WebTarget by calling a path() method when a RESTful webservice method is called then the output object will be added Response (abstract class) Object and then Response object will be return back to the client object.

→ We can read the object store in Response object by called readEntity() method

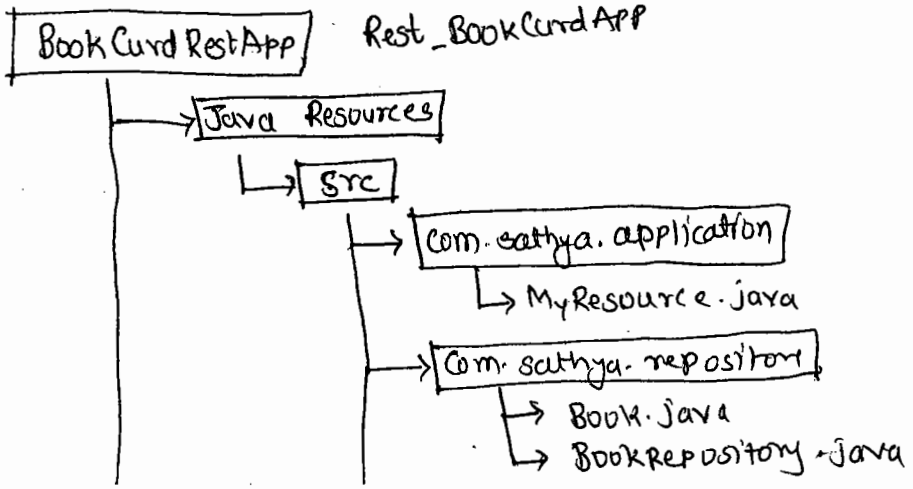
8-6-2016

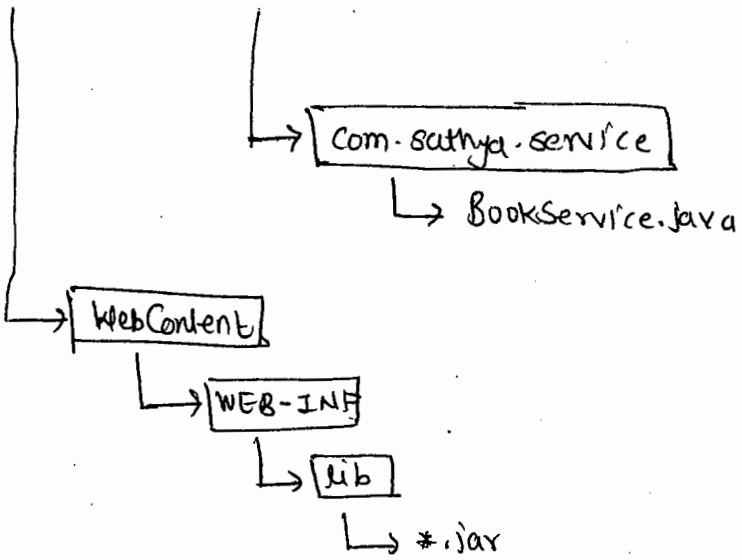
Example

→ In the following Example we are creating a Restful webservice class called BookService with 3 method.

→ BookService class takes the support of BookRepository.

→ BookService allows a client either to save a book or find a book or delete a book.





// MyResource.java

Package com.sathya.application

Import java.util.HashSet

@ApplicationPath("rest")

```
Public class MyResource extends Application
{
    private Set set;
    Public MyResource()
    {
        set = new HashSet();
        set.add(new BookServices());
    }
    Public Set getSingletons()
    {
        return set;
    }
}
```

// Book.java

```
@XmlRootElement
Public class Book implements Serializable
{
    private int bookId;
    private String bookName;
    Private double price;

Public Book () {}
Public Book (int bookId, String bookName, double price)
{
    this.bookId = bookId;
    this.bookName = bookName;
    this.price = price;
}
```

// Setters & getters.

// BookRepository.java

```
package com.sathya.repository;
import java.util.HashSet;
Public class BookRepository
{
    private Set<Book> theBooks;
    Public BookRepository ()
    {
        theBooks = new HashSet<Book>();
    }
Public Boolean saveBook (Book book)
{
```

```
boolean flag= true;
int id = book.getBookId();
If (theBooks!=null)
{
  Iterator it = theBooks.iterator();
  While (it.hasNext())
    {
      Book b = (Book) it.next();
      int bid= b.getBookId();
      if (id==bid)
        {
          flag = false;
        }
    }
}
if (flag==true)
  {
    theBooks.add(book);
    return true;
  }
 else
  {
    return false;
  }
} // end of saveBook()

Public Book getBookById (int bookId)
 {
    If (theBOOKS!=null)
     {
       Iterator it = theBooks.iterator();
       While (it.hasNext())
       {
           Book b = (Book) it.next();
```

```
        if (b.getBookId() == bookId)
         {
            return b;
         }
      }
   }
   return null;
} // end of getBookById()

Public boolean deleteBook (int bookId)
 {
    boolean flag = false;
    if (theBooks != null)
    {
       Iterator it = theBooks.iterator();
       while (it.hasNext())
        {
           Book b = (Book) it.next();
           int bid = b.getBookId();
           if (bid == bookId)
           {
              theBooks.remove(b);
              flag = true;
           }
        }
    }
   return flag;
 } // end of deleteBook()
} // end of class
```

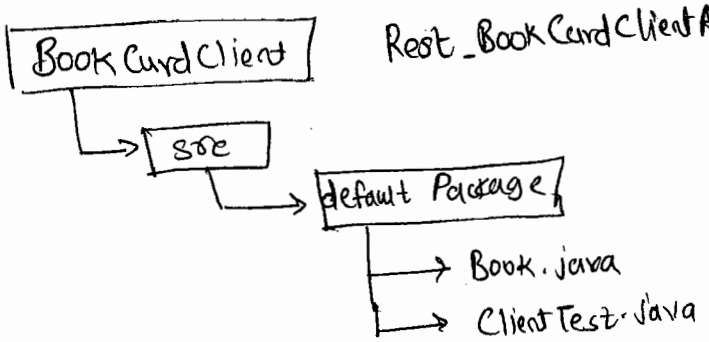
// BookService.java

```
Package com.sathya.service;

@Path ("/BookService")        // slash is optional
  Public class BookService
    {
      Private BookRepository repository = new BookRepository ();

    @ POST
    @ Path ("/save")
    @ Consumes (MediaType. APPLICATION_XML)
    @ Produces ("text/plain")

    Public String save (Book book)
     {
       boolean flag = repository.saveBOOK(book);
       if (flag)
           return "BOOK is saved";
        else
          return "Sorry! Book already exist";
     }

@GET
@Path ("/find/{id}")
@Produces (MediaType.APPLICATION_XML)
 Public Book find (@PathParam ("id") int id)
  {
     Book b = repository.getBookById (id);
     return b;
  }
```

```
@DELETE
@Path("/delete/{id}")
@Produces(MediaType.TEXT_PLAIN)
Public String delete(@PathParam("id") int id)
{
    boolean flag = repository.deleteBook(id);
    if(flag==true)
    {
        return "Book deleted";
    }
    else
    {
        return "Sorry! Book does not exist";
    }
}
```

9-6-2015

→ creating a Console Appl" as a client for the above Restful webservice




```
Rublic class ClientTest
{
    main()throws Exception
    {
        Client client = ClientBuilder.newClient();
```

```
WebTarget target = client.target("http://localhost:9898/BookCurdRestApp/
                                  rest/BookService");

// calling save operation
target = target.path("/save");
Book book = new Book(101, "Oracle", 800);
String str = (String) target.request().post(Entity.xml(book), String.class);
SOPln(str);
SOPln("=====");

// Calling find operation
WebTarget target1 = client.target("http://localhost:9898/BookCurdRestApp/
                                   rest/BookService");

target1 = target1.path("/find/{id}").resolveTemplate("id", 101);
Book book1 = (Book) target1.request().get(Book.class);
if (book1 != null)
{
    SOPln("Details of book 101");
    SOPln("book1.getBookName()");
    SOPln(book1.getPrice());
}
SOPln("====");

// Calling delete operation
WebTarget target2 = client.target("http://localhost:9898/BookCurdRestApp/rest
                                   /BookService");
target2 = target2.path("/delete/{id}").resolveTemplate("id", 103);
String str1 = (String) target2.request().delete(String.class);
    SOPln(str1);
}
}
```

O/P

```
Book is Saved
==========
Details of book 101
Oracle
800.0
==========
Sorry! Book does not exist
```

resteasy.jboss.org ⟷ downloads → Sourceforge.net

Download Resteasy jar zip. (44. MB)

→ ## Using RESTeasy

→ RESTeasy is a jboss project and it is implementation of jax-RS specification.

→ we can download RESTeasy software from resteasy.jboss.org

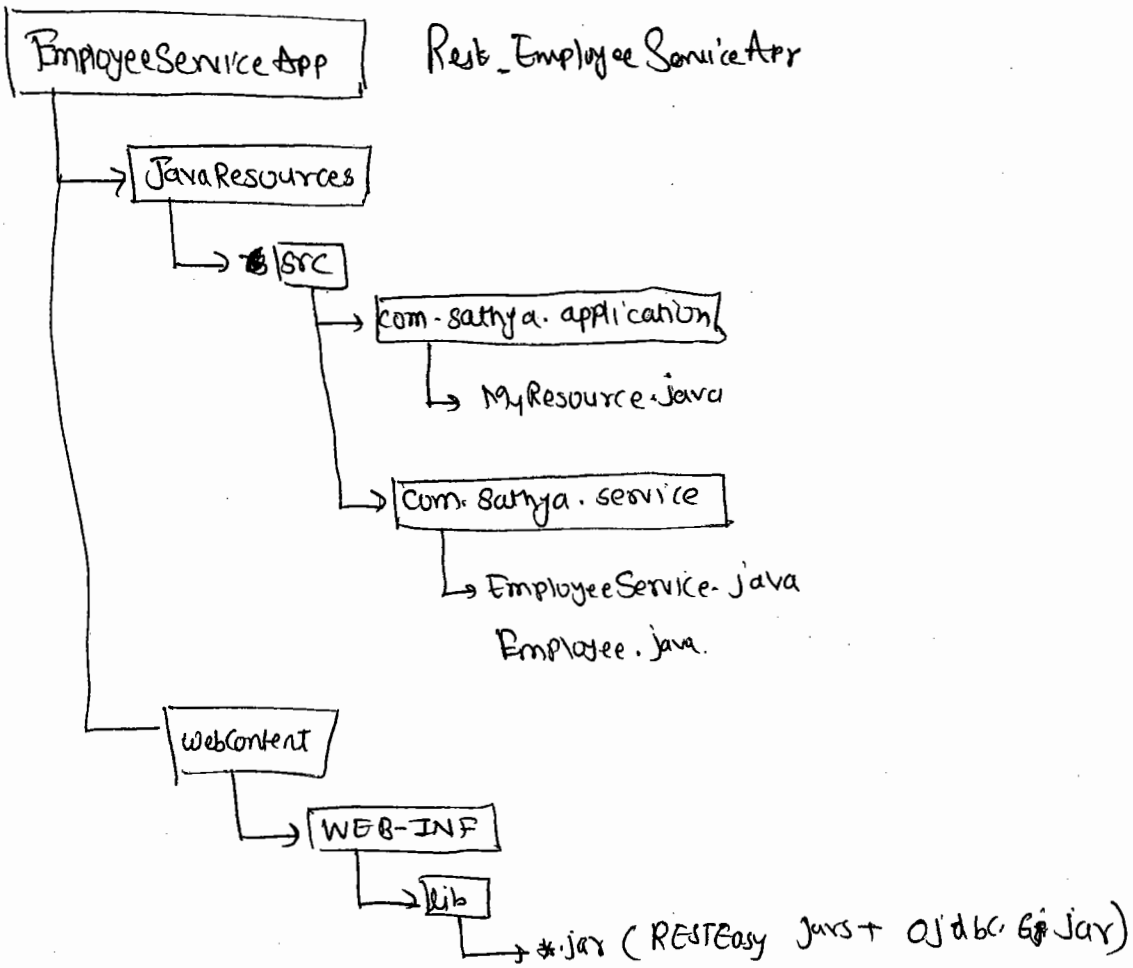
→ in restful webservices it is easy to migrate from one implementation to another implementation.

→ just we need to add a jar file to lib folder of our restful webservice application

→ a zip file with name resteasy-jaxrs-3.0.9.Final will be downloaded, hence extract zip file then a folder with the same name created

→ open lib & subfolder and copy all the jar file except resteasy-cdi-3.0.9.Final (bug in jar, remove it — two is bug jar in this jar) to the our application lib folder

⇒ In the following application we are creating a Restful Webservice class with single method which takes input as employeeId, reads data from database and returns an Employee object (POJO class)



// MyResource.java

```
@ApplicationPath("rest")
Public class MyResource extends Application
{
    private Set set;
    Public MyResource()
    {
        set = new HashSet();
        set.add(new EmployeeService());
    }
```

```
    Public Set getSingletons()
      {
        return set;
      }
  }
```

```
Public class Employee
  {
    Private int empNO;
    Private String ename;
    Private int sal;
    Private int deptno;

    // setter & getter
```

```
// EmployeeService.java

@Path("/employee")
public class EmployeeService
  {
    Private Employee e;
    Public EmployeeService(){
        e = new Employee();
    }
@Path("/search/{id}")
@GET
@Produces (MediaType.APPLICATION_JSON)
```

```
Public Employee getEmployeeById (@PathParam ("id") int id)
{
  try
    {
      Class.forName ("oracle.jdbc.OracleDriver");

Connection con = DriverManager.getConnection ("jdbc:oracle:
                                    thin:@localhost:1521:xe", "system", "");

PreparedStatement pstmt = con.PreparedStatement ("select from emp
                                                  where empNo = ?");

  pstmt.setInt (1, id);

  ResultSet rs = pstmt.executeQuery();
  if (rs.next())
     {
        e.setEmpno (rs.getInt (1));

        e.setEname (rs.getString (2));

        e.setSal (rs.getInt (3));

        e.setDeptno (rs.getInt (4));

        return e;
     }
     rs.close();
     pstmt.close();
      con.close();
  }
  Catch (Exception e)
    {
      e.printStackTrace();
    }
  return null;
}
}
```

we can send a request to the RestFul webservice method by typing following url from address bar of the browser

http://localhost:9898/EmployeeServiceApp/rest/employee/search/7788

for the above request output will be displayed in the form of json object in on the browser.

{"empno":7788, "ename":"SCOTT", "sal":6000, "deptNO":10}

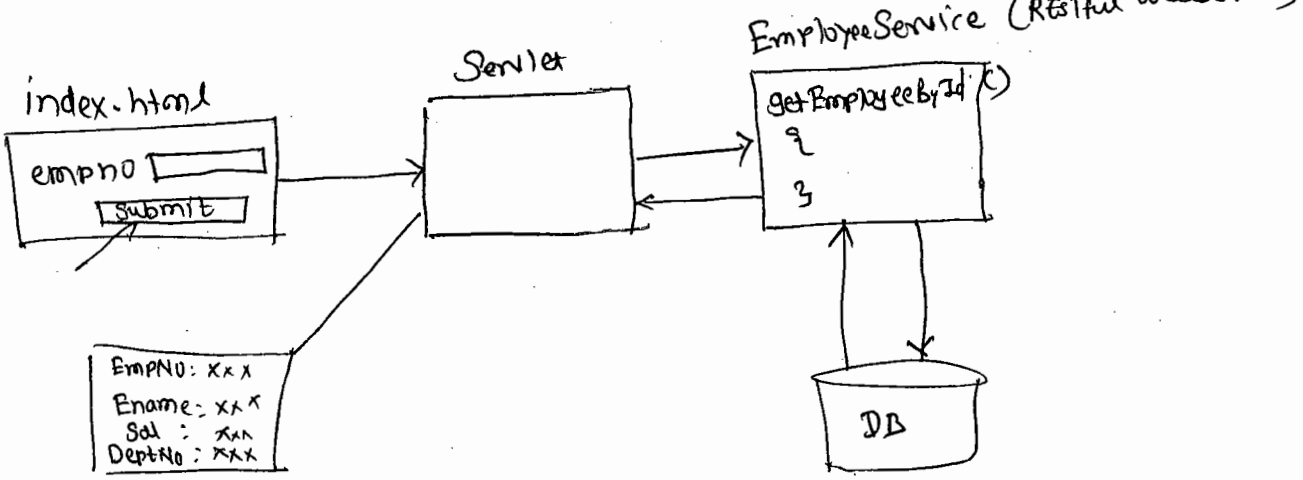
Note

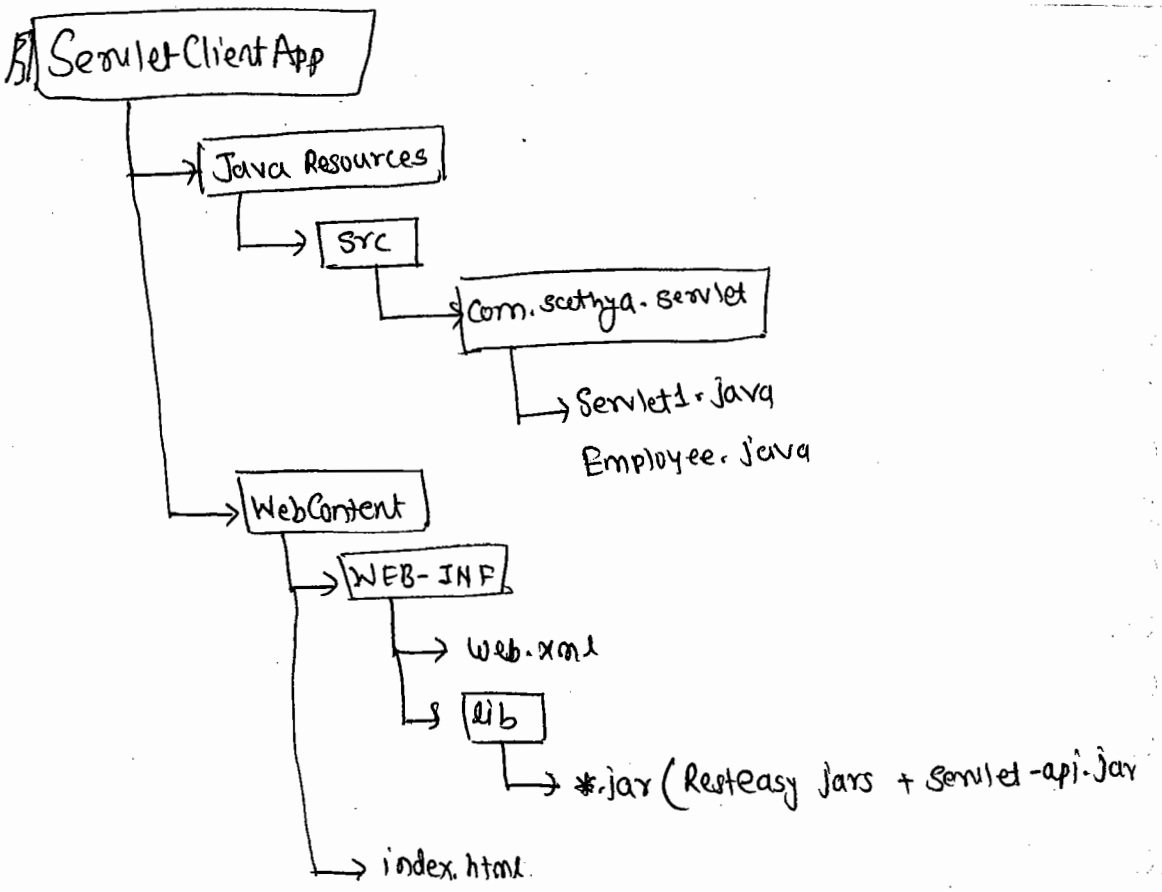
① an object json is indicated in { } symbol and object can have one or more key value pair.

② Key must be string & value can be int, double, bodean, string or object or array.

10-6-2015

⇒ In the following example we are creating a servlet as a client for calling RESTful webservice. we are sending EmployeeNo as i/p from Html to the servlet. servlet calls webservice method by passing EmployeeNo as i/p





index.html

```
<center>
  <form action = "srv1">
    Empno : <input type = text name = "tj"> <br>
            <input type = submit value = "submit">

  </form>
</center>
```

Servlet1.java

```
Public class Servlet1 extends javax.servlet.http
    {
      Protected void doGet (HttpServletRequest request, HSR respons)
```

```
{
    // read input value
    String str = request.getParameter("id");
    int eid = Integer.parseInt(str.trim());

    try
    {
        Client client = ClientBuilder.newClient();
        WebTarget target = client.target("http://localhost:2015/
                                EmployeeServiceApp/rest/employee");
        target = target.path("/search/{id}").resolveTemplate("id", eid);

        Response resp = target.request().get();
        Employee e = (Employee)resp.readEntity(Employee.class);
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        if (e!=null)
        {
            out.println("EmpNo:" +e.getEmpno());
            out.println("<br>");
            out.println("Ename :" +e.getEname());
                        <br>
                        ("sal : " + e.getSal());

                        ("Deptno:" + e.getDeptno());

        }
        else
        {
            out.println("<h2> Sorry, Empno does not exist in DB </h2>");
        }
        out.close();
    }
    catch(Exception e)
    { e.printStackTrace(); } } }
```

Exception Handling in RESTful Webservices

→ When a client calls webservice if an exception is occured in webservice then bydefault that exception will be send from Webservice to its client.

→ Webservice can be developed in one language & its client Application can developed in another language. so a client Appln cannot understand a exception.

→ It is always better to send an object with some message to the client instead of directly sending exception to the client

→ In Restful webservices, in order to map one exception to one object of a class, we need to create one ExceptionMapper class.

→ ExceptionMapper classes are created by implementing a Generic type interface called ExceptionMapper.

→ an ExceptionMapper class should be ~~annoted~~ annotated with

@Provider

outer class

```
P   C   Response
{ static ResponseBuilder ok() { }
```

return object of ResponseBuilder static inner class

```
Public Static class ResponseBuilder
{
    Response build() { }
```

inner class or nested class

← return object of Response class

```
ResponseBuilder builder = Response.ok();
Response res = builder.build();
```

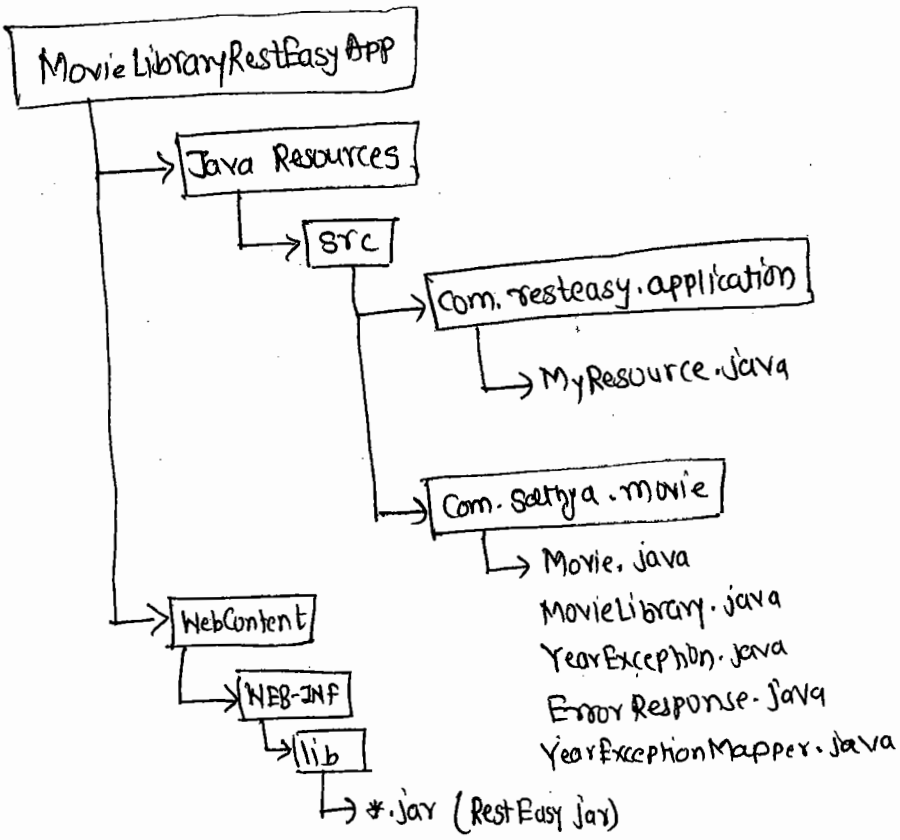
11-6-2015

→ In the following Restful Application, we are creating a webservice class MovieLibrary.

→ In MovieLibrary there is a method getMovies(), it takes input as year and returns a List of movie object, released in that year.

→ if the input year is not 2008 or 2009 an exception will be thrown.

→ We are creating an ExceptionMapper class for converting exception into an object of error response and finally error response object will be return to the client Appln



// MyResource.java

```
@ApplicationPath("rest")
Public class MyResource extends Application
{
    Private Set set;
    Public MyResource()
    {
        set = new HashSet();
        set.add(new MovieLibrary());
        set.add(new YearExceptionMapper());
    }
    Public Set getSingletons()
    {
        return set;
    }
}
```

// Movie.java

```
public class Movie
{
    Private int year;
    Private String title;
    Public Movie(int year, String title)
    {
        this.year = year;
        this.title = title;
    }
// setters & getters
```

// ErrorResponse.java

```
Public class ErrorResponse
{
    Private String message;
    // setter & getter
}
```

// YearException.java

```
Public class YearException extends RuntimeException
{
}
```

// MovieLibrary.java

```
@Path ("/movie")
Public class MovieLibrary
{
    // store the in memory movie list as 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, "Ghajini"));
                       (2008, "Jodha Akbar");
                       (2009, "Dhoom");
                       (2008, "Jalwa");
    }
```

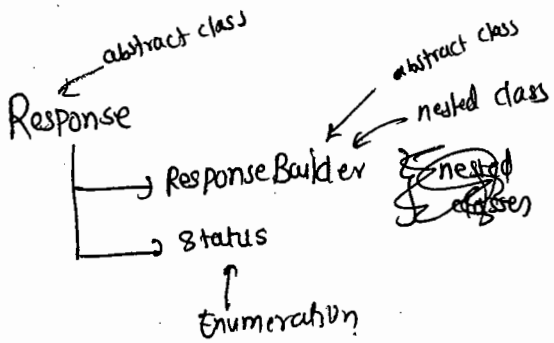
```
@GET
@Produces (MediaType. APPLICATION_JSON)
@Path ("/{year}")
Public Response getMovies (@PathParam ("year") int year) throws
                                        YearException
{
  {
    SOPln (year);
    If (year== 2008 || year == 2009)
    {
      List<Movie> list = getMoviesByYear (year);
GenericEntity<list<movie>> list2 = new GenericEntity< list <Movieo>> (list){};
      ResponseBuilder builder = Response. OK (list);
      Response response = builder. build ();
      return response;
    }
    else
      {
        throw new YearException ();
      }
  }
  Private List <movie> 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

@Provider

```
Public class YearExceptionMapper implements ExceptionMapper<YearException>
{
    @Override
    @Produces (MediaType.APPLICATION_JSON)
    Public Response toResponse (YearException e)
    {
    ErrorResponse er = new ErrorResponse ();
    er.setMessage ("Year is not valid, it must be 2008 or 2009 only");
    ResponseBuilder builder = Response.status(Response.Status.BAD_REQUEST);
    builder.entity(er);
    Response response = builder.build ();
    return response;
    }
}
```

* if enum is used in class, then at compile time it will become static, no need to create static.




```
Response.Status rs = Response.Status.BAD_REQUEST;
Response response = builder.build ();
response.status (rs);
return response;
```

→ Response is an abstract class of javax.ws.rs.core package.

→ ResponseBuilder is a nested abstract class of Response class

→ a static method of Response class is ok, returns an object of ResponseBuilder and build() method of ResponseBuilder class returns an object of Response.

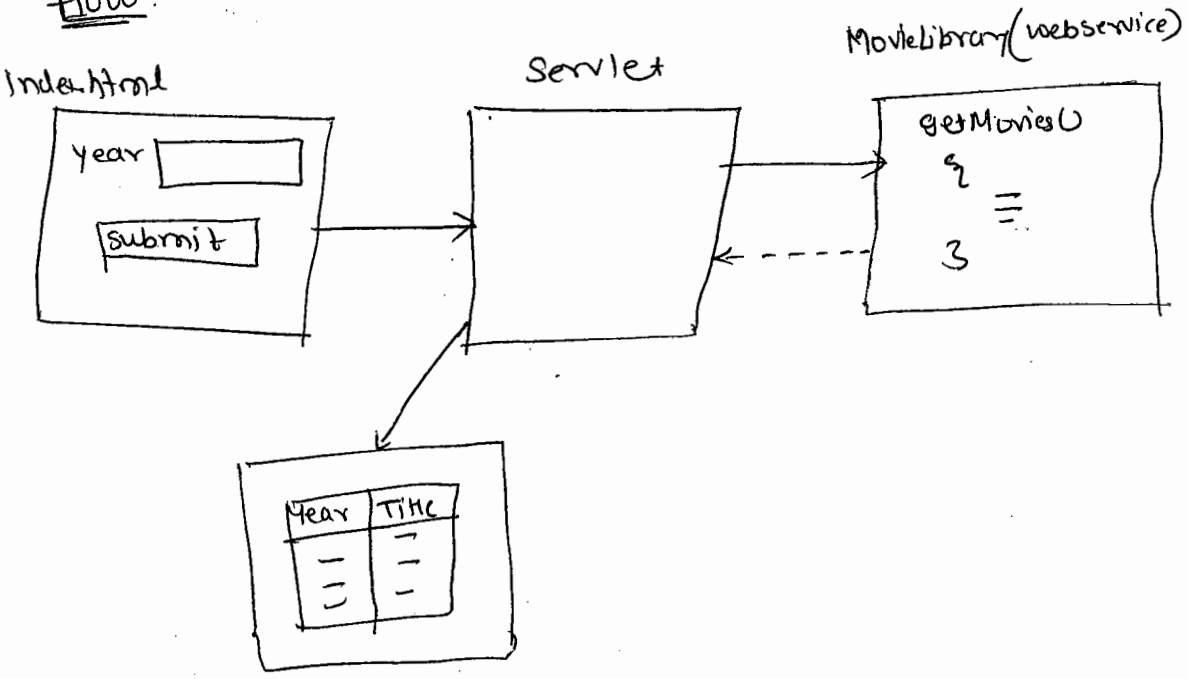
Note *** GenericType of Object added to GenericEntity object add the GenericEntity object to response object

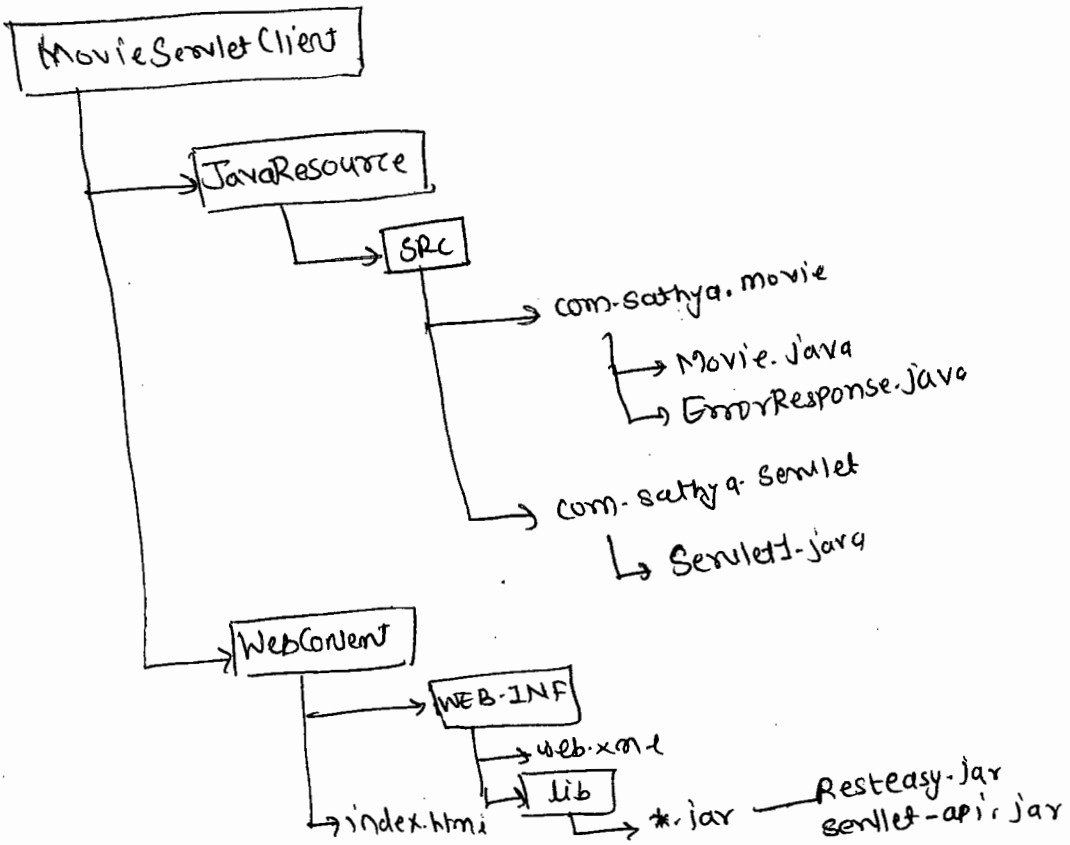
```
Public class Student
{
    Public enum Student
    {
        MALE, FEMALE;
    }
    Private int sid;
    Private String sName;
    Private Gender gender;
}
P S V. M
{
    Student s = new Student();
    S.setId();
    S.setName();
    S.setGender(Student.Gender.MALE);
```

12-6-2015

* Creating a Servlet as a client for calling above MovieLibrary webservice

flow:





// index.html

```
<center>
    <form action = "Servlet1">
        Year: <input type = text name = "t1"> <br>
            <input type = submit value = "Submit">
    </form>
</center>
```

// Servlet1.java

```
Public class Servlet1 extends HttpServlet
{
  Protected void doGet (HttpServletRequest request, HSR response)
  {
   String str = request.getParameter ("t1");
    int year = Integer.parseInt (str.trim());
  try
    {
     PrintWriter out = response.getWriter();
     response.setContentType ("text/html");

  Client client = ClientBuilder.newClient();
   WebTarget target = client.target ("http://localhost:8888/
                                    MovieLibraryRestEasyApp/rest/movie");

  target = target.path("/{year}").resolveTemplate("year", year);

  Response resp = target.request().get();
   if (resp.getStatus() != 200)
```

```
ErrorResponse er = (ErrorResponse) resp.readEntity (ErrorResponse.class);

    out.println ("<h1>" + er.getMessage () + er.getMessage () + "</h1>")
}
else
{
 GenericType <List <movie>> type = new GenericType <List <
                                   movie>>() {};


 List <Movie> list = resp.readEntity (type);

 Iterator <Movie> it = list. iterator ();

 out.println ("<center><table border = 3>");

 out.println ("<tr><th>year </th><th> Title </th></tr>");

 while (it.hasNext())
 {
    Movie movie = (Movie) it.next();
    out.println ("<tr>");
                ("<td>" + movie.getYear() + "</td>");
                ("<td>" + movie.getTitle () + "</td>");
                ("</tr>");

    }
 out.println ("</center></table>");
 out.close();
}
} catch (Exception e)
  {
   soPln(e);
  }
}
}
```

Creating Asynchronous client

→ We can create two types of client to call a restful webservice i.e Synchronous and Asynchronous

→ Synchronous client waits for a response from server. And Asynchronous client will continue its execution by without waiting for the response

→ To make a client application as a Asynchronous client Appln, we need to do following two changes

① Create a class by implementing InvocationCallback interface.

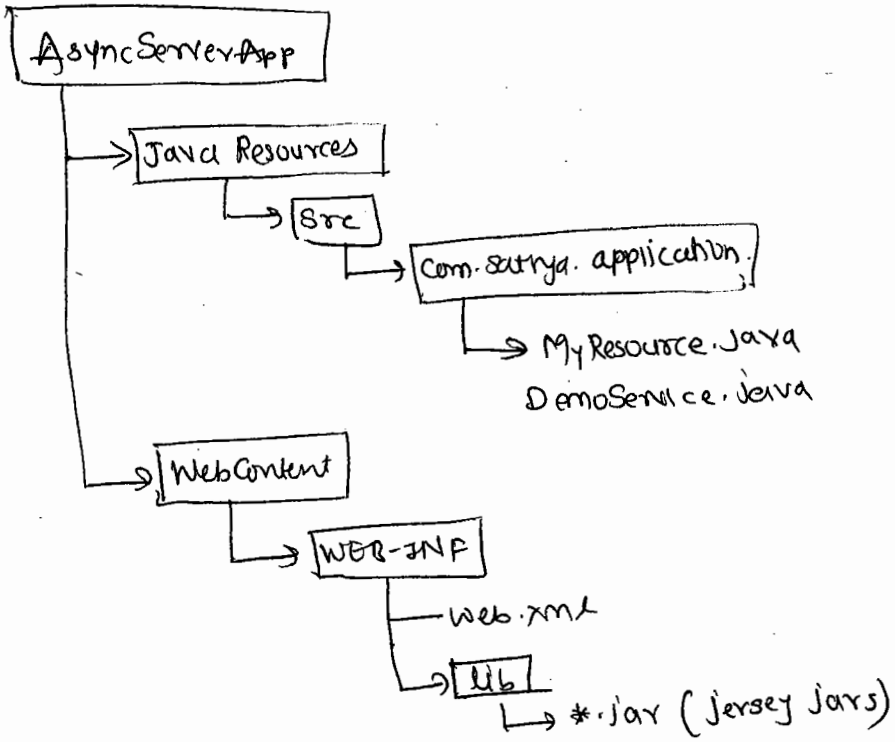
② Send request to a target Asynchronously calling async().

```
C ResponseCallback implements InvocationCallback
{
    ① completed (Response t)
    ② failed (Throwable t)
}
```

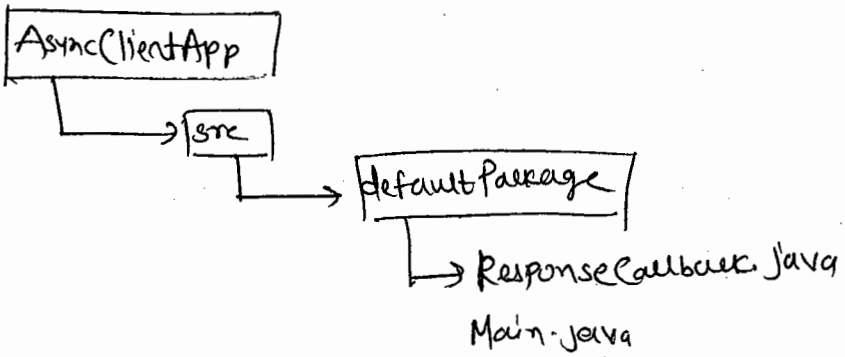
Asynchronous ⇒ target.request.async().get(new ResponseCallback());

→ In the below server-client appln we are going to test Asynchronous comm^n bet^n a browser and a server

15-6-2015,




```
// DemoService.java

@Path("/demo")

Public class DemoService
 {
    @GET
    @Path("/hello")

     Public String SayHello()
      {
          try
            {
               thread.sleep(30000);
            }
            catch (Exception e) { }
             return "Hello ....";
        }

    }
```




```
// ResponseCallback.java

Public class ResponseCallback implements InvocationCallback<Response>
{
    Public void completed (Response r)
    {
      String str = r.readEntity (String.class);
      SOPln(str);
    }
    public void failed (Throwable t)
    {

    }
}
```

```
// Main.java
  Public class Main
    {
        Main
        {
            Client client = ClientBuilder.newClient()
            WebTarget target = client.target ("http://localhost:9898/
                                              AsyncClientApp/rest/demo");

            target = target.Path ("/hello");

        AsyncInvoker ai = target.request().async();
```

```
        ai.get(new ResponseCallback());
        Sopln("I am executing")
        Sopln("I am also executing");
        Sopln("I too executing");
    }
}
```

Security in RESTful WebServices

① declarative ⟶ web.xml added security tags.

② annotations

→ In order to restrict unauthorized client to access a RESTful web service, we can provide security in 2 ways

① declarative

② annotations

Declarative Security

→ in this type of security we need to configure security tags in web.xml along with configuration of servlet given by implementation of javaEE API.

→ In web.xml, we need to configure the following security related tag

① security-constraint

- web-resource-collection
  - web-resource-name
  - url-pattern
- auth-constraint
  - role-name → group of username & password

② login-config

- auth-method.

→ a role-name indicates a group of usernames & passwords which are authorized to access a secured resource.

→ When a request comes a servlet container uses a url-pattern to find the wheather a client has send a request for a secured or non-secured resource.

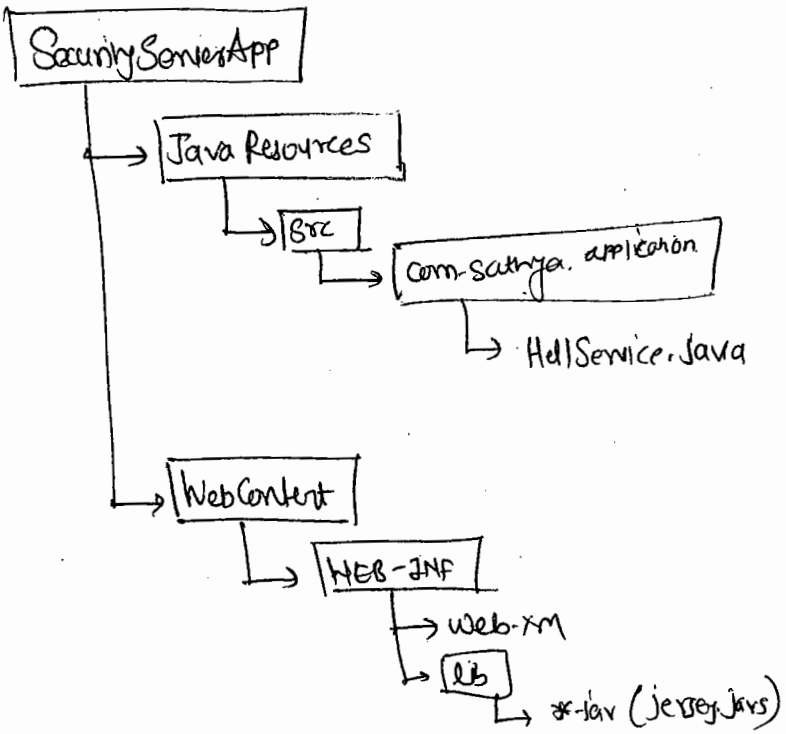
⇒ If a request comes for a secured resource and if the client is a Browser then servlet container tells browser that authentication is required.

→ a browser opens dialogue box to accept the username & password

→ In case of tomcat manager when we click ManagerApp Button in tomcat home page. Immediately browser opens a dialogue box to accept the username & Password. because manager Application is a secured resource.

16-6-2015

→ In the following example we are going to Protect our RESTful webservice in a declarative approach



// HelloService.java

```
@Path("/Hello")
Public class HelloService
{
   @GET
   @Path("/h")
   Public String sayHello()
   {
      return Hello;
   }
}
```

Web.xml

```
<web-app>
    <servlet>
        <servlet-name> jersey-servlet </servlet-name>
        <servlet-class> org.glassfish.jersey.servlet.ServletContainer </--->
        <init-param>
            <param-name> jersey.config.server.provider.packages </-->
            <param-value> com.sathy.application </param-value>
        </init-param>
        <1-0-s> 1 <1-0-s/>
    </servlet>
    <servlet-mapping>
        <servlet-name> jersey-servlet </servlet-name>
        <url-pattern> /rest/* </url-pattern>
    </servlet-mapping>
    ~~<security-cons~~
    <security-constraint>
        <web-resource-collection>
            <web-resource-name> abcd </->
            <url-pattern> /rest/hello (*) </>
        </web-resource collection>
        <auth-constraint>
            <role name> Customer </role-name>
        </auth-constraint>
    </security-constraint>
        <login-config>
            <auth-method> BASIC </auth-method>
```

</login-config>

</web-app>

→ before we deploy above app in Tomcat, we need to configure customer role in ~~tomcat-users.xml~~ tomcat-users.xml in tomcat 7 | ~~directory~~ config | directory.

```
<tomcat-users>
    <user username = "ram" password = "1234"
     roles = "customer" />
    <user username = "admin" password = "admin"
     roles = "manage-gui" />
```

We should add like this no of users.

by default

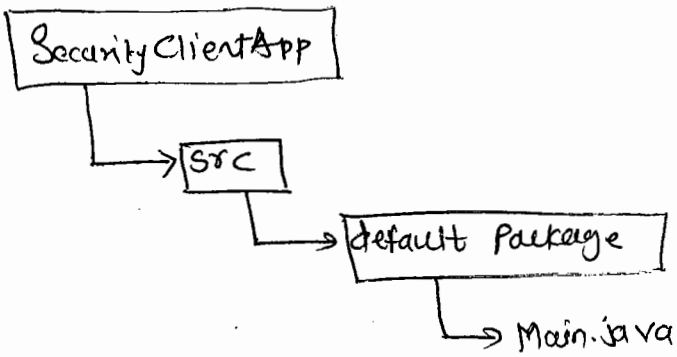
→ deploy above app in tomcat and test the application by sending a request with the following url

http://localhost:2015/securityServerApp/rest/hello/h

→ while sending a request to the Restful Webservice from the client Application, in order to send credentials (username, password) we need to the following

① create an object of HttpAuthenticationFeature class add credentials to it

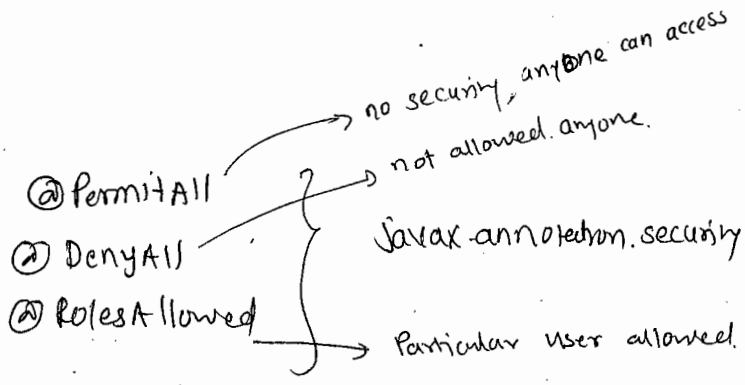
② Register ⊙ HttpAuthenticationFeature object with client object



add jar file to build path

```
Public class main
    {
      main
       {
        HttpAuthenticationFeature feature = HttpAuthenticationFeature.
        basicBuilder(). nonPreemptive(). credentials("ram", "1234").build();

      Client client = ClientBuilder.newClient();
       client.register(feature);
      WebTarget target = client.target("http://localhost:2015/SecurityServerApp
                                        /rest/hello/h");
      Response r = target.request().get();
      String str = r.readEntity(String.class);
      SOPln(str);
    }
}
```

19-6-2015



RolesAllowedDynamicFeature

↓

Class object

(Abstract class)
ResourceConfig

Security Annotations

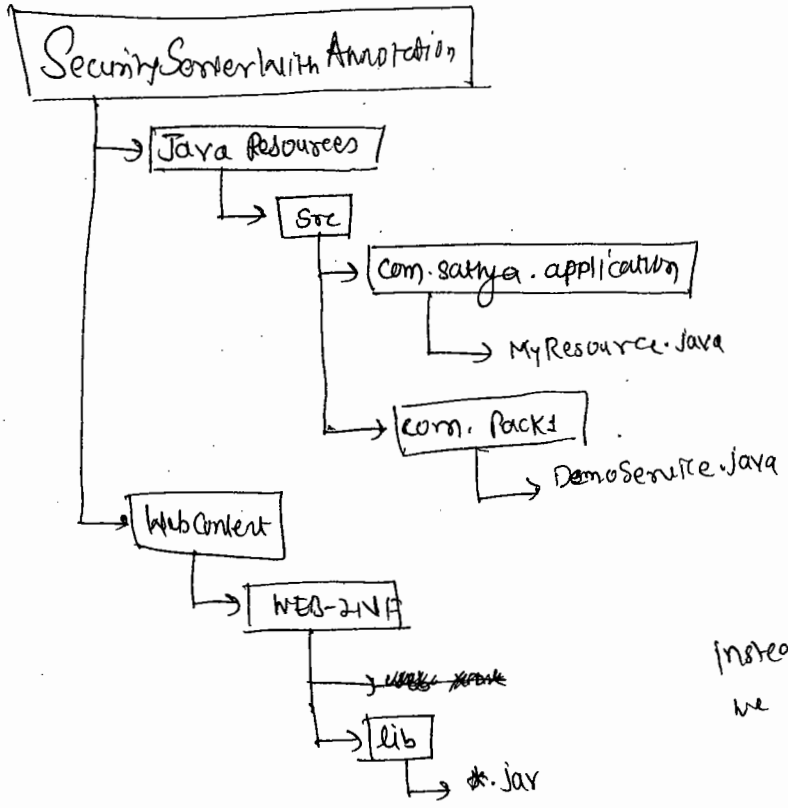
→ we can provide security to the Restful webservice using Java Security Annotations

→ 3 annotations of javax.annotation.security Package are

1. @PermitAll
2. @DenyAll
3. @RolesAllowed

→ to support the above 3 annotations, we need to register Class Object of RolesAllowedDynamicFeature with our Application

→ to register RolesAllowedDynamicFeature class we need to extend our application class from ResourceConfig abstract class. ResourceConfig is a subclass of Application class.



instead of web.xml
we are writing MyResource.java

// ~~DemoSe~~

// MyResource.java

@ApplicationPath ("rest")

```
Public class MyResource extends ResourceConfig
{
    Public MyResource()
    {
    register ( RolesAllowedDynamicFeature.class);
    Packages ("com.Pack1");
    }
}
```

```
// DemoService.java

@Path("/demo")
Public class DemoService {
   @Path("/hello")
   @PermitAll
   // @RolesAllowed ("customer");
   @Produces (MediaType.TEXT_HTML)
  @GET
   Public String sayHello()
    {
      return "<h1> Hello... </h1>";
    }

 @GET
 @DenyAll
@Path("/bye")
 public String sayBye()
   {
     return "<h1> Bye </h1>";
   }

}
```

Same Client Application is Applicable for Annotation also which is used in **SecurityClientApp** in last program

Spring Bean Make it as Restful

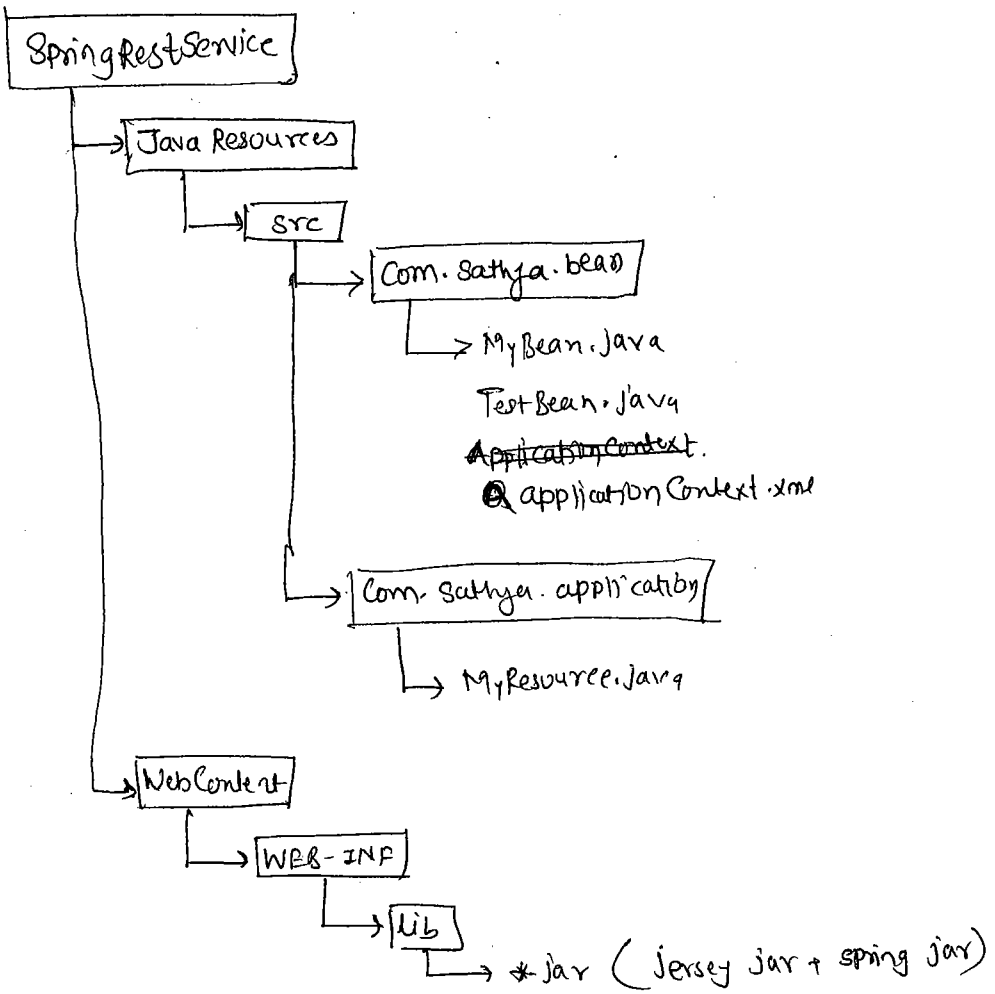
```
Public class A        <-- make it as restful
{

}
```

applicationContext.xml

```
<bean id = " "  class = " " >

</bean>
```

```
P C MyApplication extends Application
{
    P MyApplication()
    {
        Ac ctx = new CPXAC (" ");
        Object o = ctx.getBean
        ~~A a~~ = A a = (A) o;
        S.o(a);
    }
    Public Set getSingletons()
    {
        return Set;
    }
}
```

Making a SpringBean as a RESTful Service




```
Public class TestBean
{
    Public String m1()
        {
            return "Hello from Spring Bean";
        }
}
```

MyBean.java

```
@Path ("MyBean")
Public class MyBean
  {
     Private TestBean tbean;
     Public void setBean (TestBean tbean)
      {
         this.tbean = tbean;
      }
     @Path ("h")
     @GET
     Public String getService ()
      {
         String str = tbean.m1 ();
        return str;
      }
}
```

// MyResource.java

```
@ApplicationPath ("rest")
    Public class MyResource extends Application
     {
        Private Set set;
        Public MyResource ()
        {
            Set = new HashSet ();
        ApplicationContext ctx = new ClassPathXmlApplicationContext
                                 ("com/sanya/bean/applicationContext.xml");
        Object o = ctx.getBean ("myBean");
        MyBean mb = (MyBean) o;
```

```
        set.add(mb);
    }
    Public Set getSingletons()
    {
        return set;
    }
}
```

→ Deploy the above appln to the server and type the following url from the address bar

http://localhost:2015/springRestService/rest/MyBean/[illegible]h

O/P

Hello from spring bean.

applicationContext-xml

```
<beans>
<bean id="id1" class="com.satya.bean.MyBean">
    <property name="bean" ref="id2" />
</bean>
<bean id="id2" class="com.satya.bean.TestBean" />
</beans>
```