NARESH *i* technologies

XML
Web Services

Course Material - 1.0

By Mr. Ganapathi Raju

Naresh *i* Technologies, Opp. Satyam Theatre, Ameerpet, Hyd.
Office: 040- 23746666 Mobile: 9000994007, www.nareshit.com

Table of Contents

XML

(eXtensible Markup Language)

XML

Definition:

XML stands for eXtensible Markup Language and is a text-based markup language derived from Standard Generalized Markup Language (SGML).
XML is used to store and organize the data, rather than specifying how to display it like HTML tags, which are used to display the data.
The important characteristics of XML are:

- XML is extensible: XML allows you to create your own **self-descriptive** tags
- XML carries the data.
- It's not a programming a language, it's a **Markup** language.
- XML is specification of Wide Web Consortium (**W3C**) and is available as an open standard.

Markup Language: Enclosing textual content with in textual code(tags) is nothing but markup.

XML Purpose:

- XML can be used to exchange the information between organizations and systems.
- XML can be used to store and transport the data.
- XML documents are used as deployment descriptors.
- XML documents are used as textual databases.

Difference Between HTML and XML:

HTML	XML
Hiper Text Markup Language	eXtensible Markup Language
HTML was designed to display data with focus on how data looks	XML was designed to be used to transport and store data, with focus on what data is.
HTML is a markup language itself.	XML provides a framework for defining markup languages.
HTML is case insensitive.	XML is case sensitive.
HTML has it own predefined tags.	XML flexible is that custom tags.
HTML is not strict if the user does not use the closing tags.	XML makes it mandatory for the user the close each tag that has been used.
Attributes values can be present without quotation mark.	XML attribute values must be enclosed with quotations.

XML Syntax:

```
<?xml
  version="version_number"
  encoding="encoding_declaration"
  standalone="standalone_status"
 ?>
```

Parameter	Parameter value	Parameter description
Version	1.0	Specifies the version of the XML standard used.
Encoding	UTF-8, UTF-16	It defines the character encoding used in the document. UTF-8 is the default encoding used.
Standalone	yes or no.	It informs the parser whether the document relies on the information DTD/XSD. The default value is set to no

For Example:

```
<?xml version ="1.0" encoding ="UTF-8"?>
<student>
  <sno>10</sno>
  <sname>Raju</sname>
  <sage>22</sage>
</student>
```

XML Well-Formedness.

- XML Document start with PROLOG
- Start tag must have a matching end tag
- Only one Root Element
- Elements are Case sensitive
- Level Constraint.
- Attribute values must always be quoted
- Elements are case sensitive

Document PROLOG:

The document prolog comes at the top of the document, before the root element. This section contains:

- XML declaration
- Document type declaration

For Example:

XML Elements:

XML elements can be defined as building blocks of an XML.

1.Start element <elementname>

2. End element </elementname>

An element can contain

1. Other elements
2. Text
3. Attributes
4. Mix of all

XML Attributes.

Attributes are part of the XML elements. An element can have multiple unique attributes. Attribute gives more information about XML elements. To be more precise, they define properties of elements. An XML attribute is always a *name-value* pair.
If you want to add supplementary information attach to an element, instead of having another element.

- Attribute is unique in xml
- It can occur in any order.

Avoid using attributes.

- It cannot contain multiple values.
- More difficult to manipulate code.

Entities:

XML entities allow you to use text to refer to a data item, instead of using the data item itself.

You can use entities to represent:

- Characters that would otherwise cause problems for the XML processor
- Large blocks of data that need to be repeated throughout the document
- Characters that you can't type on your keyboard (i.e. ©)

Entity Reference	Character
<	<
>	>
&	&
"	"
'	'

Naming Rules:

1. Names can contain letters, numbers and other characters
2. Name cannot start with number or punctuation character.
3. Names cannot start with the letters xml
4. Names can not contain spaces

Best Practices:

1. Make names descriptive
2. Names should be short and simple
3. Avoid "-" "." ":"

XML Document:

Any text file that is developed with .xml is an xml document.

XML Application:

→Any computer app that works with an xml document is nothing but an xml application.
→Working with xml document means, Perform CRUD operations on the xml.

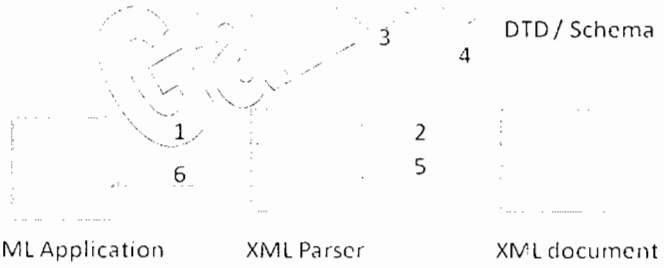
Valid XML:

If an xml document is developed according to rules specified in DTD/XSD.
→ Validating xml can be done by.
1. XML Parsers
2. XML Editors

XML Parser:

Its an API that enables XML applications to work with xml document.
It performs the following operations:
1.Reading the xml document
2.Checking the well-formedness
3.Verify its validity.
4.Making XML data available to XML application.

Diagrammatical Representation:



1.XML application instantiating the parser and specifying the XML file to the parser.
2.Parser reads the specified XML document and verify its well-formedness
3.In the xml document parser gets the information about DTD or schema.
4.Parser reads the metadata specified in the DTD or schema.
5.Basing on the metadata read into memory.
6.Parser makes XML document data available to the XML application either in the form of tree structure or in the form of events

DTD

DTD stands for Document Type Definition. A DTD allows you to create rules for the elements within your XML documents.
If XML file holds data, its corresponding DTD holds meta data.
In a DTD legal building blocks of an XML documents are specified.
DTDs check the validity of, structure of an XML document against the grammatical rules of the appropriate XML language.
An XML document can be defined as:

- **Well-formed XML**: If the XML document adheres to all the general XML rules such as tags must be properly nested, opening and closing tags must be balanced, and empty tags must end with '/>', then it is called as well-formed.

- **Valid XML**: An XML document said to be valid when it is not only well-formed, but it also conforms to available DTD/XSD that specifies which tags it uses, what attributes those tags can contain, and which tags can occur inside other tags, among other properties.

Features of DTD:

Following are some important points that a DTD describes:

- The elements that can appear in an XML document.
- The order in which they can appear.
- Optional and mandatory elements.
- Element attributes and whether they are optional or mandatory.
- Whether attributes can have default values.

How to associate a DTD with XML:

There are two types of association.
1. Internal DTD.
2. External DTD.

1. Internal DTD

A DTD is referred to as an internal DTD if elements are declared within the XML files. To reference it as internal DTD, standalone attribute in XML declaration must be set to yes. This means the declaration works independent of external source.

Syntax:
The syntax of internal DTD is as shown:

<!DOCTYPE root-element [element-declarations]>

Where root-element is the name of root element and element-declarations is where you declare the elements.

Example:
Following is a simple example of internal DTD:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<!DOCTYPE student [
  <!ELEMENT student (sno,sname,sage)>
  <!ELEMENT sno (#PCDATA)>
  <!ELEMENT sname (#PCDATA)>
```

```
  <!ELEMENT sage (#PCDATA)>
]>
<student>
  <sno>001</sno>
  <sname>Ganapathi Raju</sname>
  <sage>22</sage>
</student>
```

Let us go through the above code:

XML declaration with following statement

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

DTD:

<!DOCTYPE student [

The DOCTYPE declaration has an exclamation mark (!) at the start of the element name. The DOCTYPE informs the parser that a DTD is associated with this XML document.

DTD Body- The DOCTYPE declaration is followed by body of the DTD, where you declare elements, attributes, entities, and notations:

```
<!ELEMENT student (sno,sname,sage)>
  <!ELEMENT sno (#PCDATA)>
  <!ELEMENT sname (#PCDATA)>
  <!ELEMENT sage (#PCDATA)>
```

Here #PCDATA means parse-able text data(Parsed character Data).

End Declaration - Finally, the declaration section of the DTD is closed using a closing bracket and a closing angle bracket (]>). This effectively ends the definition, and thereafter, the XML document follows immediately.

2. External DTD

In external DTD elements are declared outside the XML file. They are accessed by specifying the system attributes which may be either the legal .dtd file or a valid URL. To reference it as external DTD, standalone attribute in the XML declaration must be set as no. This means, declaration includes information from the external source.

Syntax:

Following is the syntax for external DTD:

<!DOCTYPE root-element SYSTEM/PUBLIC "file-name">

Example:

The following example shows external DTD usage:

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<!DOCTYPE student SYSTEM "student.dtd">
<student>
 <sno>001 </sno>
 <sname> Ganapathi Raju </sname>
 <sage>22</sage>
</student>
```

The content of the DTD file student.dtd are as shown:

```
<!ELEMENT student (sno,sname,sage)>
 <!ELEMENT sno (#PCDATA)>
 <!ELEMENT sname (#PCDATA)>
 <!ELEMENT sage (#PCDATA)>
```

Types:

You can refer to an external DTD by either using.

1.System identifiers

2.Public identifiers.

System Identifiers:

A system identifier enables you to specify the location of an external file containing DTD declarations. Syntax is as follows:

```
<!DOCTYPE student SYSTEM "student.dtd" [...]>
```

As you can see it contains keyword SYSTEM and a URI reference pointing to the location of the document.

Public Identifiers:

When declaring a DTD available for public use, you need to use the PUBLIC keyword within your DOCTYPE declaration. When you use the PUBLIC keyword, you also need to use an *FPI* (which stands for Formal Public Identifier).

FPI Syntax:

An FPI is made up of 4 fields, each separated by double forward slashes (//):

```
field 1//field 2//field 3//field 4
```

FPI Example:

Here's a real life example of an FPI. In this case, the DTD was created by the Sun Microsystems for web.xml:

<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.2//EN"
"http://java.sun.com/j2ee/dtds/web-app_2_2.dtd">

1. web-app = root element
2. **PUBLIC**
   The Availability. The most common DOCTYPES you will use will be publicly available - "PUBLIC". But you can also specify a local DTD with the "SYSTEM" key word.
3. FPI Fields

An FPI must contain the following fields:

Field	Example	Description
Separator	//	This is used to separate the different fields of the FPI.
First field	-	Indicates whether the DTD is connected to a formal standard or not. If the DTD hasn't been approved (for example, you've defined the DTD yourself), use a hyphen (-). If the DTD has been approved by a nonstandard body, use a plus sign "+". If the DTD has been approved by a formal standards body this field should be a reference to the standard itself.
Second field	Sun Microsystems, Inc	Holds the name of the group (or person) responsible for the DTD. The above example is maintained by the Sun Microsystems, Inc. so " Sun Microsystems, Inc " appears in the second field.
Third field	DTD Web Application 2.2	Indicates the type of document that is being described. This usually contains some form of unique identifier (such as a version number).
Fourth field	EN	Specifies the language that the DTD uses. This is achieved by using the two letter identifier for the language (i.e. for English, use "EN").

XML Editors:

1. Altova XML Spy
2. Eclipse . etc.....

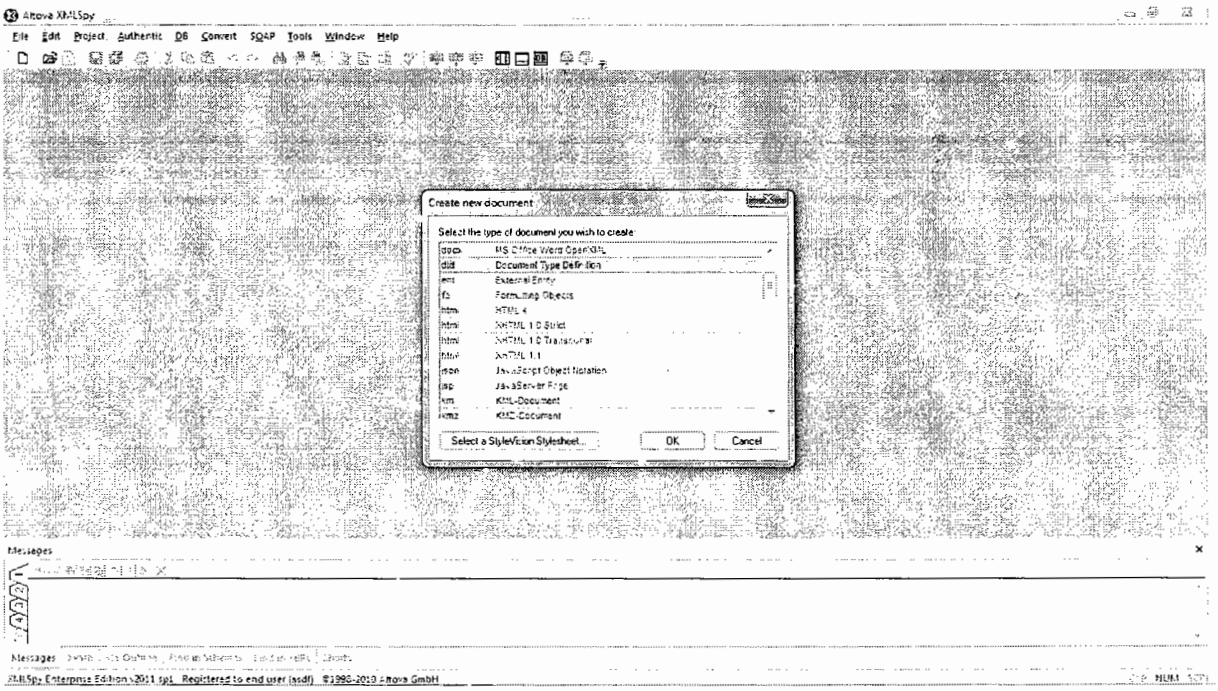
Altova XML Spy:

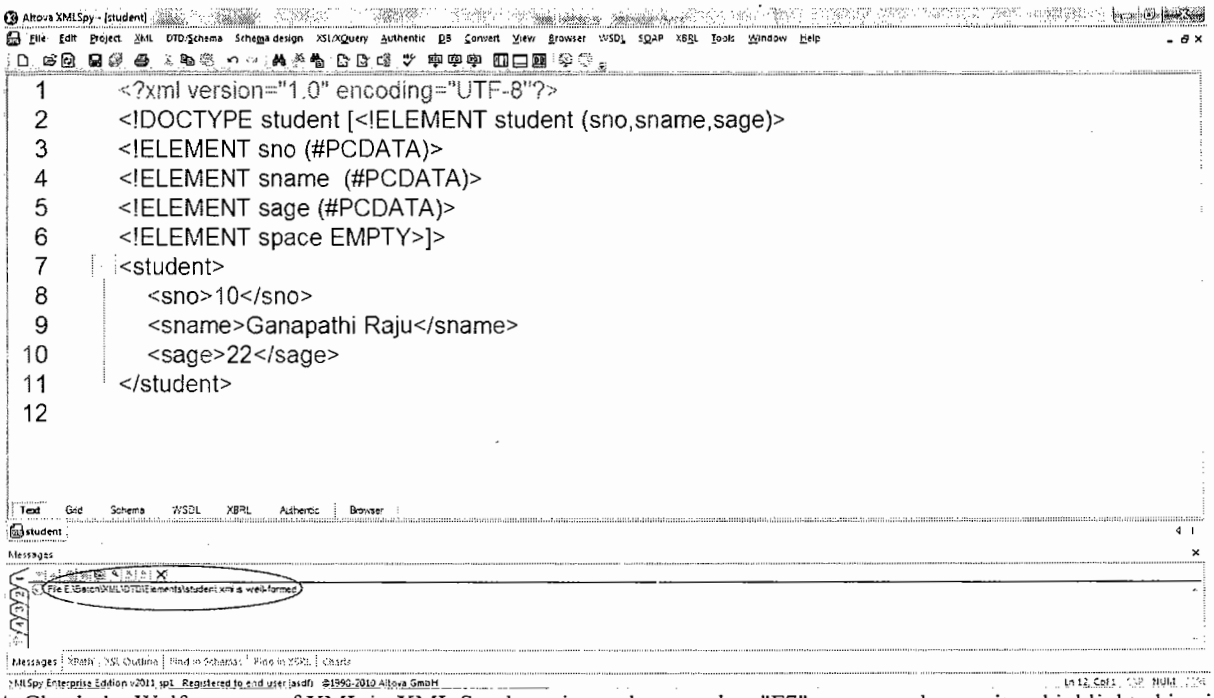
Altova XML Spy is best XML editor with powerful support for working with all XML-based technologies like XML, XSD,XSLT ,XQUERY,WSDL and so on.

The advanced functionality in XML Spy is coupled with user-friendly views and entry helpers, wizards, and debuggers designed to help you create, edit, and optimize XML-based applications.

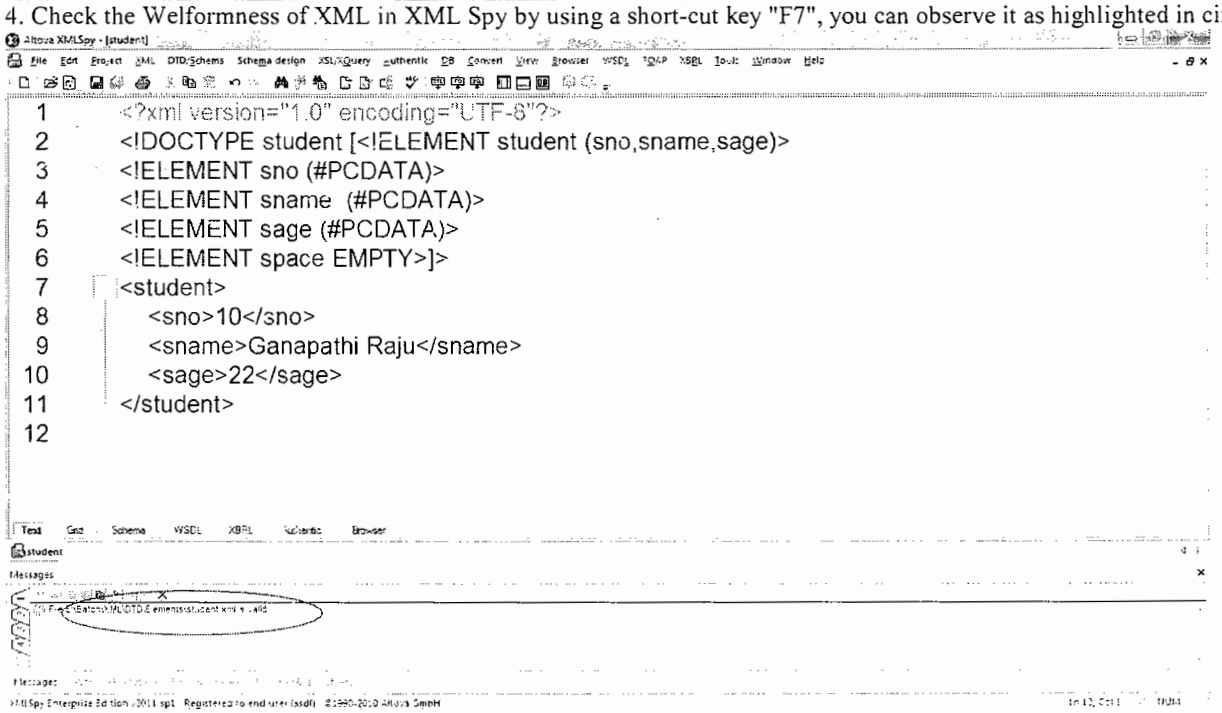
Steps to work with XML Spy:

1. Download and install Altova XML Spy:
2. Choose to create DTD or XML from File-->New-->DTD or XML
3 . Write a DTD and XML file as shown in below figure.





4. Check the Welformness of XML in XML Spy by using a short-cut key "F7", you can observe it as highlighted in cirlce.



5. Verify the validity of XML in XML Spy by using a short-cut key "F8", you can observe it as high lighted in cirlce.

The Building Blocks of XML Documents:

A DTD will basically contain declarations of the following XML components:

- Elements
- Attributes
- Entities
- PCDATA
- CDATA

1.Elements:

XML elements can be defined as building blocks of an XML document. Elements can behave as a container to hold text, elements, attributes or mix of all.

To define an element in your DTD, you use the <!ELEMENT> declaration. The actual contents of your <!ELEMENT> declaration will depend on the syntax rules you need to apply to your element.

The <!ELEMENT> declaration has the following syntax:

```
<!ELEMENT element_name content_model>
```

Here, element_name is the name of the element you're defining.
The content model could indicate a specific rule, data or another element.

- If it specifies a rule, it will be set to either ANY or EMPTY.
- If specifies data or another element, the data type/element name needs to be surrounded by brackets (i.e. (student) or (#PCDATA)).

The following examples show you how to use this syntax for defining your elements.

1.Plain Text:

If an element should contain plain text, you define the element using #PCDATA. PCDATA stands for Parsed Character Data and is the way you specify non-markup text in your DTDs.

Example -

<sname>Ganapathi Raju</sname>

The "Ganapathi Raju" part is the PCDATA. The other part consists of markup.

Syntax:

```
<!ELEMENT element_name (#PCDATA)>
```

Example:

```
<!ELEMENT sname (#PCDATA)>
```

2. Unrestricted Elements:

If it doesn't matter what your element contains, you can create an element using the content model of ANY. Note: This removes all syntax checking, so you should avoid using this if possible. You're better off defining a specific content model.

Syntax:

```
<!ELEMENT element_name ANY>
```

Example:

```
<!ELEMENT student ANY>
```

3.Empty Elements:

We might remember that an empty element is one without a closing tag. For example, in XHTML, the
 and <img /> tags are empty elements. Here's how you define an empty element:

Syntax:

```
<!ELEMENT element_name EMPTY>
```

Example:

```
<!ELEMENT br EMPTY>
```

The above line in your DTD defines the following empty element for your XML document:

```
<br/>
```

4.Child Elements:

We can specify that an element must contain another element, by providing the name of the element it must contain. Here's how you do that:

Syntax:

```
<!ELEMENT element_name (child_element_name)>
```

Example:

```
<!ELEMENT student (sno)>
```

The above line in your DTD allows the "student" element to contain one instance of the "sname" element in your XML document:

```
<student>
  <sname>Ganapathi Raju</sname>
</student>
```

5.Multiple Child Elements (Sequences):

We can also provide a comma separated list of elements if it needs to contain more than one element. This is referred to as a "sequence". The XML document must contain the tags in the same order that they're specified in the sequence.

Syntax:

```
<!ELEMENT element_name (child_element_name, child_element_name,...)>
```

Example:

```
<!ELEMENT student(sname, sage)>
```

The above line in your DTD allows the "student" element to contain one instance of the "sname" element and one instance of the "sage" element in your XML document:

```
<student>
    <sname>Ganapathi Raju</sname>
    <sage>22</sage>
 </student>
```

Cardinally Operators:

It indicates the no. of occurrences of elements that can be presented in the document.

This is fine if there only needs one instance of "student", but what if we didn't want a limit. What if the " college" element should be able to contain any number of "student " instances? Fortunately we can do that using DTD operators.

Here's a list of operators/syntax rules we can use when defining child elements:

Operator	Syntax	Description
+	a+	One or more occurences of **a**
*	a*	Zero or more occurences of **a**
?	a?	Either **a** or nothing
,	a, b	**a** followed by **b**
|	a | b	**a** followed by **b**
()	(expression)	An expression surrounded by parentheses is treated as a unit and could have any one of the following suffixes ?, *, or +.

Zero or More
To allow zero or more of the same child element, use an asterisk (*):

Syntax:
```
<!ELEMENT element_name (child_element_name*)>
```

Example:
```
<!ELEMENT college (student*)>
```

one or more
To allow one or more of the same child element, use a plus sign (+):

Syntax:
```
<!ELEMENT element_name (child_element_name+)>
```

Example:
```
<!ELEMENT college (student+)>
```

Zero or One
To allow either zero or one of the same child element, use a question mark (?):

Syntax:
```
<!ELEMENT element_name (child_element_name?)>
```

Example:
```
<!ELEMENT college (student?)>
```

Choices:
You can define a choice between one or another element by using the pipe (|) operator. For example, if the "student" element requires a child called either "sno", "sname", or "sage" (but only one of these), you can do the following:

Syntax:
```
<!ELEMENT element_name (choice_1 | choice_2 | choice_3)>
```

Example:
```
<!ELEMENT student (sno | sname | sage)>
```

Mixed Content:
You can use the pipe (|) operator to specify that an element can contain both PCDATA and other elements:

Syntax:
```
<!ELEMENT element_name (#PCDATA | child_element_name)>
```

Example:

```
<!ELEMENT student (#PCDATA | sno | sname | sage)*>
```

2. Attributes:

Attributes are part of the XML elements. An element can have any number of unique attributes. Attributes give more information about the XML element or more precisely it defines a property of the element. An XML attribute is always a name-value pair.
You use a single <!ATTLIST> declaration to declare all attributes for a given element. In other words, for each element (that contains attributes), you only need one <!ATTLIST> declaration.
The <!ATTLIST> declaration has the following syntax:

```
<!ATTLIST element-name attribute-name attribute-type attribute-value>

DTD example:

<!ATTLIST student gender CDATA "male">

XML example:

<student gender="male" />
```

The attribute TYPE field can be set to one of the following values:

Value	A simple text value, enclosed in quotes.
#IMPLIED	Specifies that there is no default value for this attribute, and that the attribute is optional.
#REQUIRED	There is no default value for this attribute, but a value must be assigned.
#FIXED value	The #FIXED part specifies that the value must be the value provided. The value part represents the actual value.

A Default Attribute Value

```
DTD:
<!ELEMENT square EMPTY>
<!ATTLIST square width CDATA "0">

Valid XML:
<square width="100" />
```

In the example above, the "square" element is defined to be an empty element with a "width" attribute of type CDATA. If no width is specified, it has a default value of 0.

#REQUIRED

Syntax:

```
<!ATTLIST element-name attribute-name attribute-type #REQUIRED>
```

Example

```
DTD:
<!ATTLIST student gender CDATA #REQUIRED>
Valid XML:
<student gender="male" />

Invalid XML:
<student/>
```

Use the #REQUIRED keyword if you don't have an option for a default value, but still want to force the attribute to be present.

#IMPLIED

Syntax

```
<!ATTLIST element-name attribute-name attribute-type #IMPLIED>
```

Example

```
DTD:
<!ATTLIST student gender CDATA #IMPLIED>

Valid XML:
<student gender="male" />

Valid XML:
<student />
```

Use the #IMPLIED keyword if you don't want to force the author to include an attribute, and you don't have an option for a default value.

#FIXED

Syntax

```
<!ATTLIST element-name attribute-name attribute-type #FIXED "value">
```

Example

```
DTD:
<!ATTLIST student technology CDATA #FIXED "Java">
Valid XML:
<student technology ="Java" />
Invalid XML:
<student technology =".Net" />
```

Use the #FIXED keyword when you want an attribute to have a fixed value without allowing the author to change it. If an author includes another value, the XML parser will return an error.

Enumerated Attribute Values:

Syntax

```
<!ATTLIST element-name attribute-name (en1|en2|..) default-value>
```

Example

```
DTD:
<!ATTLIST student gender (male|female) "male">
XML example:
<student gender ="male" />
or
<student gender ="female" />
```

Use enumerated attribute values when you want the attribute value to be one of a fixed set of legal values.

The attribute TYPE field can be set to one of the following values:

Type	**Description**
CDATA	Character Data (text that doesn't contain markup)
ENTITY	The name of an entity (which must be declared in the DTD)
ENTITIES	A list of entity names, separated by whitespaces. (All entities must be declared in the DTD)
Enumerated	A list of values. The value of the attribute must be one from this list.
ID	A unique ID or name. Must be a valid XML name.
IDREF	Represents the value of an ID attribute of another element.
IDREFS	Represents multiple IDs of elements, separated by whitespace.
NMTOKEN	A valid XML name.
NMTOKENS	A list of valid XML names, separated by whitespace.
NOTATION	A notation name (which must be declared in the DTD).

CDATA

Example:

```
<!ATTLIST student sname CDATA "Ganapathi Raju">
```

Valid XML - The following XML document would be valid, as it conforms to the above DTD:

```
<college>
 <student sname="Ganapthi Raju">
 </student>
 <student sname="Krishnam Raju">
 </student>
</college>
```

3.Entities:

1. used to define shortcuts
2. it can be declared internal
3. it can be declared external

Internal:

<!ENTITY entity-name "entity-value">

External:

<!ENTITY entity-name SYSTEM "URI/URL">

4.CDATA:

The term CDATA means, Character Data. CDATA are defined as blocks of text that are not parsed by the parser, but are otherwise recognized as markup.

The predefined entities such as <, >, and & require typing and are generally difficult to read in the markup. In such cases, CDATA section can be used. By using CDATA section, you are commanding the parser that the particular section of the document contains no markup and should be treated as regular text.

Following is the syntax for CDATA section:

```
<![CDATA[
   characters with markup
]]>
```

The above syntax is composed of three sections:

- **CDATA Start section** - CDATA begins with the nine-character delimiter <![CDATA[
- **CDATA End section** - CDATA section ends with]]> delimiter.
- **CData section** - Characters between these two enclosures are interpreted as characters, and not as markup. This section may contain markup characters (<, >, and &), but they are ignored by the XML processor.

Example

The following markup code shows example of CDATA. Here, each character written inside the CDATA section is ignored by the parser.

```
<query>
     <![CDATA[ select * from student where sno < 10 ]]>
 </query>
```

In the above syntax, everything between <query > and </query > is treated as character data and **not** as markup.

Advantages of using DTD:

- **Documentation** - You can define your own format for the XML files. Looking at this document a user/developer can understand the structure of the data.
- **Validation** - It gives a way to check the validity of XML files by checking whether the elements appear in the right order, mandatory elements and attributes are in place, the elements and attributes have not been inserted in an incorrect way, and so on.
- With a DTD, each of your XML files can carry a description of its own format.
- With a DTD, independent groups of people can agree to use a standard DTD for interchanging data.
- Your application can use a standard DTD to verify that the data you receive from the outside world is valid.
- You can also use a DTD to verify your own data.

Disadvantages of using DTD:

- It supports only the #PCDATA data type. i.e weaker in data types.
- Doesn't follow XML syntax and hence not productive.
- Cardinally Operator can't be effectively specified.
- It does not support the namespaces. Namespace is a mechanism by which element and attribute names can be assigned to groups. However, in a DTD namespaces have to be defined within the DTD, which violates the purpose of using namespaces.

XSD(XML Schema Definition)

XSD:

XML Schema Definition commonly known as XSD is a way to describe precisely the XML language. XSDs check the validity of, structure and vocabulary of an XML document against the grammatical rules of the appropriate XML language.

Features:

- XSDs can be extensible for future additions.
- XSD is richer and more powerful than DTD.
- XSD is written in XML.
- XSD supports data types
- XSD supports namespaces
- XSD is W3C recommendation
- Schemas are another approach used by the XML parser to validate an XML document

Schema Defines:

- Elements appear in a document
- Attributes appear in a document
- No. of elements occurrence in a document.
- Empty elements or include text
- Data types of elements and attributes
- Default and fixed values of an elements and attributes

Create an XML Schema:

Now we want to create a schema for the XML document above.

We start by opening a new file that we will call "student.xsd".

To create the schema we could simply follow the structure in the XML document and define each element as we find it.

We will start with the standard XML declaration followed by the xs:schema element that defines a schema:

```
<?xml version="1.0" encoding="UTF-8" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
...
</xs:schema>
```

In the schema above we use the standard namespace (xs), and the URI associated with this namespace is the Schema language definition, which has the standard value of http://www.w3.org/2001/XMLSchema.

```
<?xml version="1.0"?>

<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.rajuhelps2u.com"
xmlns="http://www.rajuhelps2u.com"
elementFormDefault="qualified">
...
...
</xs:schema>
```

The following fragment:

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

Indicates that the elements and data types used in the schema come from the "http://www.w3.org/2001/XMLSchema" namespace. It also specifies that the elements and data types that come from the "http://www.w3.org/2001/XMLSchema" namespace should be prefixed with xs:

This fragment:

```
targetNamespace="http://www.rajuhelps2u.com"
```

indicates that the elements defined by this schema (sno, sname) come from the "http://www.rajuhelps2u.com" namespace.

This fragment:

```
xmlns="http://www.rajuhelps2u.com"
```

indicates that the default namespace is "http://www.rajuhelps2u.com".

This fragment:

```
elementFormDefault="qualified"
```

Indicates that any elements used by the XML instance document which were declared in this schema must be namespace qualified.

Referencing a Schema in an XML Document:

This XML document has a reference to an XML Schema:

```
<?xml version="1.0"?>
<student xmlns="http://www.rajuhelps2u.com"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.rajuhelps2u.com student.xsd">
<sno>001</sno>
<sname>Ganapathi</sname>
</student>
```

The following fragment:

```
xmlns="http://www.rajuhelps2u.com"
```

Specifies the default namespace declaration. This declaration tells the schema-validator that all the elements used in this XML document are declared in the "http://www.rajuhelps2u.com" namespace.

Once you have the XML Schema Instance namespace available:

```
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

you can use the schemaLocation attribute. This attribute has two values, separated by a space. The first value is the namespace to use. The second value is the location of the XML schema to use for that namespace:

```
xsi:schemaLocation="http://www.rajuhelps2u.com student.xsd"
```

XML Schemas define the elements of your XML files.

- Simple Elements.
- Complex Elements.

Simple Elements :

A simple element is an XML element that contains only text. It cannot contain any other elements or attributes. However, the "only text" restriction is quite misleading. The text can be of many different types. It can be one of the types included in the XML Schema definition (boolean, string, date, etc.), or it can be a custom type that you can define yourself.

Defining a Simple Element:

The syntax for defining a simple element is:

```
<xs:element name="name_of_element" type="type_of_element"/>
```

where "name_of_element" is the name of the element and "type_of_element" is the data type of the element.

XML Schema has a lot of built-in data types. The most common types are:

- xs:string
- xs:decimal
- xs:integer
- xs:boolean
- xs:date
- xs:time

Example:

```
<sname>Raju</sname>
<sage>22</sage>
<dob>04-12-1985</dob>
```

And here are the corresponding simple element definitions:

```
<?xml version="1.0" encoding="UTF-8" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
<xs:element name="sname" type="xs:string"/>
<xs:element name="sage" type="xs:integer"/>
<xs:element name="dob" type="xs:date"/>
</xs:schema>
```

Default and Fixed values for elements:

Simple elements may have a default value OR a fixed value specified.

A default value is automatically assigned to the element when no other value is specified.

In the following example the default value is "raju":

```
<xs:element name="sname" type="xs:string" default="raju"/>
```

A fixed value is also automatically assigned to the element, and you cannot specify another value.

In the following example the fixed value is "red":

```
<xs:element name="sname" type="xs:string" fixed="raju"/>
```

XSD Attribute:

Simple elements cannot have attributes. If an element has attributes, it is considered to be of a complex type. But the

attribute itself is always declared as a simple type.

How to Define an Attribute?
The syntax for defining an attribute is:

```
<xs:attribute name="name of attr" type="type of attr"/>
```

where name_of attr is the name of the attribute and type_of_attr specifies the data type of the attribute.

XML Schema has a lot of built-in data types. The most common types are:

- xs:string
- xs:decimal
- xs:integer
- xs:boolean
- xs:date
- xs:time

Example:
Here is an XML element with an attribute:

```
<sname gender="male">Raju</sname>
```

And here is the corresponding attribute definition:

```
<xs:attribute name="gender" type="xs:string"/>
```

Default and Fixed Values for Attributes:

Attributes may have a default value OR a fixed value specified.

A default value is automatically assigned to the attribute when no other value is specified.

In the following example the default value is "male":

```
<xs:attribute name="gender" type="xs:string" default="male"/>
```

A fixed value is also automatically assigned to the attribute, and you cannot specify another value.

In the following example the fixed value is "male":

```
<xs:attribute name="gender" type="xs:string" fixed="male"/>
```

Optional and Required Attributes

Attributes are optional by default. To specify that the attribute is required, use the "use" attribute:

```
<xs:attribute name="gender" type="xs:string" use="required"/>
```

XML Restrictions:

Restrictions are used to define acceptable values for XML elements or attributes. Restrictions on XML elements are called facets.

Restrictions on Values:

The following example defines an element called "sage" with a restriction. The value of age cannot be lower than 18 or greater than 60.

```
<xs:element name="sage">
  <xs:simpleType>
    <xs:restriction base="xs:integer">
      <xs:minInclusive value="18"/>
      <xs:maxInclusive value="60"/>
    </xs:restriction>
```

```
  </xs:simpleType>
</xs:element>
```

Restrictions on a Set of Values:

To limit the content of an XML element to a set of acceptable values, we would use the enumeration constraint.
The example below defines an element called "course" with a restriction.
The only acceptable values are: Java, .Net, Oracle:

```
<xs:element name="coure">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:enumeration value="Java"/>
      <xs:enumeration value=".Net"/>
      <xs:enumeration value="Oracle"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

Restrictions on a Series of Values:

To limit the content of an XML element to define a series of numbers or letters that can be used, we would use the pattern constraint.
The example below defines an element called "name" with a restriction. The only acceptable value is ONE of the LOWERCASE letters from a to z:

```
<xs:element name="name">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:pattern value="[a-z]"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

The next example defines an element called "name" with a restriction. The only acceptable value is THREE of the UPPERCASE letters from a to z:

```
<xs:element name="name">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:pattern value="[A-Z][A-Z][A-Z]"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

The next example also defines an element called "name" with a restriction. The only acceptable value is THREE of the LOWERCASE OR UPPERCASE letters from a to z:

```
<xs:element name="name">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:pattern value="[a-zA-Z][a-zA-Z][a-zA-Z]"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

The next example defines an element called "choice" with a restriction. The only acceptable value is ONE of the following letters: x, y, OR z:

```
<xs:element name="choice">
  <xs:simpleType>
    <xs:restriction base="xs:string">
```

```
      <xs:pattern value="[xyz]"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

The next example defines an element called "id" with a restriction. The only acceptable value is FIVE digits in a sequence, and each digit must be in a range from 0 to 9:

```
<xs:element name="id">
  <xs:simpleType>
    <xs:restriction base="xs:integer">
      <xs:pattern value="[0-9][0-9][0-9][0-9][0-9]"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

Other Restrictions on a Series of Values:

The example below defines an element called "name" with a restriction. The acceptable value is zero or more occurrences of lowercase letters from a to z:

```
<xs:element name="name">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:pattern value="([a-z])*"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

The next example also defines an element called "letter" with a restriction. The acceptable value is one or more pairs of letters, each pair consisting of a lower case letter followed by an upper case letter. For example, "rAjU" will be validated by this pattern, but not "Raju" or "RAJU" or "raju":

```
<xs:element name="name">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:pattern value="([a-z][A-Z])+"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

The next example defines an element called "gender" with a restriction. The only acceptable value is male OR female:

```
<xs:element name="gender">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:pattern value="male|female"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

The next example defines an element called "password" with a restriction. There must be exactly eight characters in a row and those characters must be lowercase or uppercase letters from a to z, or a number from 0 to 9:

```
<xs:element name="password">
  <xs:simpleType>
```

```
    <xs:restriction base="xs:string">
      <xs:pattern value="[a-zA-Z0-9]{8}"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

Restrictions on Whitespace Characters:

To specify how whitespace characters should be handled, we would use the whiteSpace constraint.
This example defines an element called "address" with a restriction. The whiteSpace constraint is set to "preserve", which means that the XML processor WILL NOT remove any white space characters:

```
<xs:element name="address">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:whiteSpace value="preserve"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

This example also defines an element called "address" with a restriction. The whiteSpace constraint is set to "replace", which means that the XML processor WILL REPLACE all white space characters (line feeds, tabs, spaces, and carriage returns) with spaces:

```
<xs:element name="address">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:whiteSpace value="replace"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

This example also defines an element called "address" with a restriction. The whiteSpace constraint is set to "collapse", which means that the XML processor WILL REMOVE all white space characters (line feeds, tabs, spaces, carriage returns are replaced with spaces, leading and trailing spaces are removed, and multiple spaces are reduced to a single space):

```
<xs:element name="address">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:whiteSpace value="collapse"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

Restrictions on Length:

To limit the length of a value in an element, we would use the length, maxLength, and minLength constraints.
This example defines an element called "password" with a restriction. The value must be exactly eight characters:

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

This example defines another element called "password" with a restriction. The value must be minimum five characters and

maximum eight characters:

```
<xs:element name="password">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:minLength value="5"/>
      <xs:maxLength value="8"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

Restrictions for Datatypes:

Constraint	Description
enumeration	Defines a list of acceptable values
fractionDigits	Specifies the maximum number of decimal places allowed. Must be equal to or greater than zero
length	Specifies the exact number of characters or list items allowed. Must be equal to or greater than zero
maxExclusive	Specifies the upper bounds for numeric values (the value must be less than this value)
maxInclusive	Specifies the upper bounds for numeric values (the value must be less than or equal to this value)
maxLength	Specifies the maximum number of characters or list items allowed. Must be equal to or greater than zero
minExclusive	Specifies the lower bounds for numeric values (the value must be greater than this value)
minInclusive	Specifies the lower bounds for numeric values (the value must be greater than or equal to this value)
minLength	Specifies the minimum number of characters or list items allowed. Must be equal to or greater than zero
pattern	Defines the exact sequence of characters that are acceptable
totalDigits	Specifies the exact number of digits allowed. Must be greater than zero
whiteSpace	Specifies how white space (line feeds, tabs, spaces, and carriage returns) is handled

XSD Complex Type:

A complex element contains other elements and/or attributes.
There are four kinds of complex elements:

- Empty elements
- Elements that contain only other elements
- Elements that contain only text
- Elements that contain both other elements and text

Note: Each of these elements may contain attributes as well!

Examples of Complex Elements:

A complex XML element, "student", which is empty:

```
<student sid="001"/>
```

A complex XML element, "student", which contains only other elements:

```
<student>
  <sname>Raju</sname>
  <sage>22</sage>
</student>
```

A complex XML element, "student", which contains only text:

```
<studen id="001">Raju</student>
```

A complex XML element, "language", which contains both elements and text:

```
<lanquage>
I am learning java from <date>01.01.2006</date> ....
</lanquage>
```

How to Define a Complex Element:

Look at this complex XML element, "student", which contains only other elements:

```
<student>
  <firstname>Ganapathi</firstname>
  <lastname>Raju</lastname>
</student>
```

We can define a complex element in an XML Schema two different ways:

1. The "student" element can be declared directly by naming the element, like this:

```
<xs:element name="student">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="firstname" type="xs:string"/>
      <xs:element name="lastname" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

If you use the method described above, only the "student" element can use the specified complex type. Note that the child elements, "firstname" and "lastname", are surrounded by the <sequence> indicator. This means that the child elements must appear in the same order as they are declared. You will learn more about indicators in the XSD Indicators chapter.

2. The "student" element can have a type attribute that refers to the name of the complex type to use:

```
<xs:element name="student" type="name"/>

<xs:complexType name="name">
  <xs:sequence>
    <xs:element name="firstname" type="xs:string"/>
    <xs:element name="lastname" type="xs:string"/>
  </xs:sequence>
</xs:complexType>
```

If you use the method described above, several elements can refer to the same complex type, like this:

```
<xs:element name="employee" type="name"/>
<xs:element name="student" type="name"/>
<xs:element name="member" type="name"/>

<xs:complexType name="name">
  <xs:sequence>
    <xs:element name="firstname" type="xs:string"/>
    <xs:element name="lastname" type="xs:string"/>
  </xs:sequence>
</xs:complexType>
```

Complex Empty Elements:

An empty XML element:

```
<student sid="001"/>
```

The "student" element above has no content at all. To define a type with no content, we must define a type that allows elements in its content, but we do not actually declare any elements, like this:

```
<xs:element name="student">
  <xs:complexType>
    <xs:complexContent>
      <xs:restriction base="xs:integer">
        <xs:attribute name="sid" type="xs:positiveInteger"/>
      </xs:restriction>
    </xs:complexContent>
  </xs:complexType>
</xs:element>
```

Complet Types Containing Elements Only:

An XML element, "student", that contains only other elements:

```
<student>
  <firstname>Ganapathi</firstname>
  <lastname>Raju</lastname>
</student>
```

You can define the "student" element in a schema, like this:

```
<xs:element name="student">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="firstname" type="xs:string"/>
```

```
      <xs:element name="lastname" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Complex Types with Mixed Content:
An XML element, "letter", that contains both text and other elements:

```
<instituite>
  Dear Mr.<name>Ganapathi Raju</name>.
  Your id is: <sid>1032</sid>
  will be joined on <date>2001-07-13</date>.
</instituite>
```

The following schema declares the "institute" element:

```
<xs:element name="institute">
  <xs:complexType mixed="true">
    <xs:sequence>
      <xs:element name="name" type="xs:string"/>
      <xs:element name="sid" type="xs:positiveInteger"/>
      <xs:element name="date" type="xs:date"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

XSD Complex Indicators:

Indicators controls the ways how elements are to be organized in an XML document.
There are seven types of indicators falling in three broad categories.

- **Order Indicators**
  - **All** - Child elements can occur in any order.
  - **Choice** - Only one of the child element can occur.
  - **Sequence** - Child element can occur only in specified order.
- **Occurence Indicators**
  - **maxOccurs** - Child element can occur only maxOccurs number of times.
  - **minOccurs** - Child element must occur minOccurs number of times.
- **Group Indicators**
  - **Group** - Defines related set of elements.
  - **attributeGroup** - Defines related set of attributes.

Order Indicators:
Using **<all>** a student element can have firstname, lastname and marks child element in any order in the XML document.

```
<xs:complexType name="StudentType">
   <xs:all>
      <xs:element name="firstname" type="xs:string"/>
      <xs:element name="lastname" type="xs:string"/>
      <xs:element name="marks" type="xs:positiveInteger"/>
   </xs:all>
</xs:complexType>
<xs:element name='student' type='StudentType' />
```

using **<choice>** a student element can have only one of firstname, lastname and marks child element in the XML document.

```
<xs:complexType name="StudentType">
   <xs:choice>
      <xs:element name="firstname" type="xs:string"/>
      <xs:element name="lastname" type="xs:string"/>
      <xs:element name="marks" type="xs:positiveInteger"/>
   </xs:choice>
</xs:complexType>

<xs:element name='student' type='StudentType' />
```

Using <sequence> a student element can have firstname, lastname and marks child element in the specified order only in the XML document.

```
<xs:complexType name="StudentType">
   <xs:sequence>
      <xs:element name="firstname" type="xs:string"/>
      <xs:element name="lastname" type="xs:string"/>
      <xs:element name="marks" type="xs:positiveInteger"/>
   </xs:sequence>
</xs:complexType>
<xs:element name='student' type='StudentType' />
```

Occurence Indicators:

Using <maxOccurs> a student element can have maximum two mobiles in the XML document.

```
<xs:element name='student'
<xs:complexType>
   <xs:all>
      <xs:element name="name" type="xs:string"/>
      <xs:element name="mobile" type="xs:string" maxOccurs="2"/>
   </xs:all>
</xs:complexType>
</xs:element>
```

Using <minOccurs> a student element should have two mobiles in the XML document.

```
<xs:element name='student'
<xs:complexType>
   <xs:all>
      <xs:element name="name" type="xs:string"/>
      <xs:element name="mobile" type="xs:string" maxOccurs="2"/>
   </xs:all>
</xs:complexType>
</xs:element>
```

Group Indicators:

<group> is used to group related set of element. Here we've created a group of part of name and then used this group to define a student element.

```
<xs:group name="infogroup">
  <xs:sequence>
    <xs:element name="firstname" type="xs:string"/>
    <xs:element name="lastname" type="xs:string"/>
   </xs:sequence>
</xs:group>

<xs:element name="student">
```

```
<xs:complexType>
   <xs:sequence>
     <xs:group ref="infogroup"/>
     <xs:element name="marks" type="xs:integer"/>
   </xs:sequence>
</xs:complexType>
</xs:element>
```

<attributeGroup> is used to group related set of attribute. Here we've created a group of part of name and then used this group to define attributes for student element.

```
<xs:attributeGroup name="infogroup">
    <xs:attribute name="firstname" type="xs:string"/>
    <xs:attribute name="lastname" type="xs:string"/>
</xs:attributeGroup>

<xs:element name="student">
<xs:complexType>
   <xs:sequence>
     <xs:attributeGroup ref="infogroup"/>
     <xs:element name="marks" type="xs:integer"/>
   </xs:sequence>
</xs:complexType>
</xs:element>
```

The <any> Element:

The <any> element enables us to extend the XML document with elements not specified by the schema.

The following example is a fragment from an XML schema called "student.xsd". It shows a declaration for the "student" element. By using the <any> element we can extend (after <lastname>) the content of "student" with any element:

```
<xs:element name="student">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="firstname" type="xs:string"/>
      <xs:element name="lastname" type="xs:string"/>
      <xs:any minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Now we want to extend the "student" element with a "address" element. In this case we can do so, even if the author of the schema above never declared any "address" element.

Look at this schema file, called "address.xsd":

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.rajuhelps2u.com"
xmlns="http://www.rajuhelps2u.com"
elementFormDefault="qualified">

<xs:element name="address">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="city" type="xs:string"
```

```
      maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

</xs:schema>
```

The XML file below "Students.xml" uses components from two different schemas; "student.xsd" and "address.xsd":

```
<?xml version="1.0" encoding="UTF-8"?>
<students xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.naresh.com student.xsd
http://www.rajuhelps2u.com address.xsd">
<student>
  <firstname>Ganapathi</firstname>
  <lastname>Raju</lastname>
  <address>
    <city>Bhimavaram</city>
  </address>
</student>

<student>
  <firstname>Krishnam</firstname>
  <lastname>Raju</lastname>
</student>
</students>
```

The XML file above is valid because the schema "student.xsd" allows us to extend the "student" element with an optional element after the "lastname" element.

The <any> and <anyAttribute> elements are used to make EXTENSIBLE documents! They allow documents to contain additional elements that are not declared in the main XML schema.

The <anyAttribute> Element:

The <anyAttribute> element enables us to extend the XML document with attributes not specified by the schema.

The following example is a fragment from an XML schema called "student.xsd". It shows a declaration for the "student" element. By using the <anyAttribute> element we can add any number of attributes to the "student" element:

```
<xs:element name="student">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="firstname" type="xs:string"/>
      <xs:element name="lastname" type="xs:string"/>
    </xs:sequence>
    <xs:anyAttribute/>
  </xs:complexType>
</xs:element>
```

Now we want to extend the "student" element with a "gender" attribute. In this case we can do so, even if the author of the schema above never declared any "gender" attribute.

Look at this schema file, called "attribute.xsd":

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

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.rajuhelps2u.com"
xmlns="http://www.rajuhelps2u.com"
elementFormDefault="qualified">

<xs:attribute name="gender">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:pattern value="male|female"/>
    </xs:restriction>
  </xs:simpleType>
</xs:attribute>
</xs:schema>
```

The XML file below "Students.xml" uses components from two different schemas; "student.xsd" and "attribute.xsd":

```
<?xml version="1.0" encoding="UTF-8"?>
<students xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:SchemaLocation="http://www.naresh.com student.xsd
http://www.rajuhelps2u.com attribute.xsd">
<student gender="female">
  <firstname>Hege</firstname>
  <lastname>Refsnes</lastname>
</student>
<student gender="male">
  <firstname>Stale</firstname>
  <lastname>Refsnes</lastname>
</student>
</students>
```

The XML file above is valid because the schema "student.xsd" allows us to add an attribute to the "student" element.

The below example to cover all the restrictions and order indicators.

1.employee.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<!--W3C Schema generated by XMLSpy v2011 sp1 (http://www.altova.com)-->
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
      <xs:element name="eno" type="xs:integer"/>
      <xs:element name="ename">
            <xs:simpleType>
                  <xs:restriction base="xs:string">
                        <xs:minLength value="4"/>
                        <xs:maxLength value="9"/>
                  </xs:restriction>
            </xs:simpleType>
      </xs:element>
      <xs:element name="eage">
            <xs:simpleType>
                  <xs:restriction base="xs:integer">
                        <xs:minExclusive value="18"/>
                        <xs:maxExclusive value="60"/>
                  </xs:restriction>
            </xs:simpleType>
      </xs:element>
```

```
	<xs:element name="phone">
		<xs:simpleType>
			<xs:restriction base="xs:integer">
				<xs:totalDigits value="10"/>
			</xs:restriction>
		</xs:simpleType>
	</xs:element>
	<xs:element name="salary">
		<xs:simpleType>
			<xs:restriction base="xs:decimal">
				<xs:totalDigits value="6"/>
				<xs:fractionDigits value="2"/>
			</xs:restriction>
		</xs:simpleType>
	</xs:element>
	<xs:element name="email">
		<xs:simpleType>
			<xs:restriction base="xs:string">
				<xs:pattern value="[a-z0-9]*@gmail.com"/>
			</xs:restriction>
		</xs:simpleType>
	</xs:element>
	<xs:element name="jobtype">
		<xs:complexType>
			<xs:choice>
				<xs:element name="govt"/>
				<xs:element name="private"/>
			</xs:choice>
		</xs:complexType>
	</xs:element>
	<xs:element name="eadress">
		<xs:complexType>
			<xs:sequence>
				<xs:element name="city">
					<xs:simpleType>
						<xs:restriction base="xs:string">
							<xs:pattern
value="Hyderabad|Secunderabad"/>
						</xs:restriction>
					</xs:simpleType>
				</xs:element>
				<xs:element name="state">
					<xs:simpleType>
						<xs:restriction base="xs:string">
							<xs:enumeration value="Andhra"/>
							<xs:enumeration value="Tamilnadu"/>
							<xs:enumeration value="Karnataka"/>
							<xs:enumeration value="Telangana"/>
						</xs:restriction>
					</xs:simpleType>
				</xs:element>
			</xs:sequence>
		</xs:complexType>
	</xs:element>
	<xs:element name="employee">
		<xs:complexType>
			<xs:all>
				<xs:element ref="eno"/>
				<xs:element ref="ename"/>
```

```
                        <xs:element ref="eage"/>
                        <xs:element ref="eadress"/>
                        <xs:element ref="phone"/>
                        <xs:element ref="salary"/>
                        <xs:element ref="email"/>
                        <xs:element ref="jobtype"/>
                    </xs:all>
                </xs:complexType>
        </xs:element>
</xs:schema>
```

2.employee.xml (Valid XML)

```
<?xml version="1.0" encoding="UTF-8"?>
<employee xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="file:///E:/Batch/XML/XSD/employee.xsd">
        <eno>20</eno>
        <ename>Ganapathi</ename>
        <eage>22</eage>
        <eadress>
                <city>Hyderabad</city>
                <state>Andhra</state>
        </eadress>
        <phone>1234567890</phone>
        <salary>1234.50</salary>
        <email>rajuhelps2u@gmail.com</email>
        <jobtype>
                <govt>SBI Clerk</govt>
        </jobtype>
</employee>
```

3.employee.xml (Not Valid XML)

```
<?xml version="1.0" encoding="UTF-8"?>
<employee xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="file:///E:/Batch/XML/XSD/employee.xsd">
        <eno>10</eno>
        <ename>Ganapathi Raju</ename>
        <eage>22</eage>
        <eadress>
                <city>Hyderabad</city>
                <state>Andhra</state>
        </eadress>
        <phone>12345678901234</phone>
        <salary>1234.1234</salary>
        <email>rajuhelps2u@mymail.com</email>
        <jobtype>
                <govt>SBI Clerk</govt>
        </jobtype>
</employee>
```

The above employee.xml is not valid because of following reason.

- "ename" contains more than 9 characters.
- "phone" contains more than 10 digits.
- "email" is not belongs to gmail.com
- "salary" is having more than 2 fractional digits.

The below example to cover all the occurrence indicator and group indicators.

1.student.xsd

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

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
	<xs:element name="student">
		<xs:complexType>
			<xs:sequence>
				<xs:element name="sno" type="xs:integer" fixed="10"/>
				<xs:group ref="namegroup"/>
				<xs:element name="marks">
					<xs:complexType>
						<xs:attributeGroup ref="subjectgroup"/>
					</xs:complexType>
				</xs:element>
				<xs:element name="address" type="addressType"/>
			</xs:sequence>
		</xs:complexType>
	</xs:element>
	<xs:complexType name="addressType">
		<xs:sequence>
			<xs:element name="city" type="xs:string"/>
			<xs:element name="state" type="xs:string" default="A.P"/>
			<xs:element name="mobile" type="xs:integer" maxOccurs="2"
minOccurs="1"/>
		</xs:sequence>
	</xs:complexType>
	<xs:group name="namegroup">
		<xs:sequence>
			<xs:element name="firstname" type="xs:string"/>
			<xs:element name="lastname" type="xs:string"/>
		</xs:sequence>
	</xs:group>
	<xs:attributeGroup name="subjectgroup">
		<xs:attribute name="english" type="xs:string"/>
		<xs:attribute name="maths" type="xs:string"/>
	</xs:attributeGroup>
</xs:schema>
```

2.student.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<student xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="file:///E:/Batch/XML/XSD/student.xsd">
	<sno>10</sno>
	<firstname>Ganapathi</firstname>
	<lastname>Raju</lastname>
	<marks english="10" maths="20"/>
	<address>
		<city>Hyd</city>
		<state>A.P</state>
		<mobile>123</mobile>
	</address>
</student>
```

Difference Between DTD and XSD?

DTD	XSD
Document Type Definition	XML Schema Definition
DTD's are not XML type document	XSD's are XML type documents
DTD's are tough to learn new syntax (EBNF-Extensible Backup Norm Form).	XML type documents are easy to learn.
Limited data types. So not type safe (#PCDATA)	Rich in data types
Cardinality control over the elements is limited	Much more Cardinally Operator
DTD's are don't allow to create own data types.	Allow to create user-defined data types using complex type
Doesn't support name spaces	Support name spaces

Why XSD?

1. Its supports different data types.
2. XML schema uses xml syntax only.
3. XML schema secure data communication.
4. XML schemas are extensible.

XML Name Space:

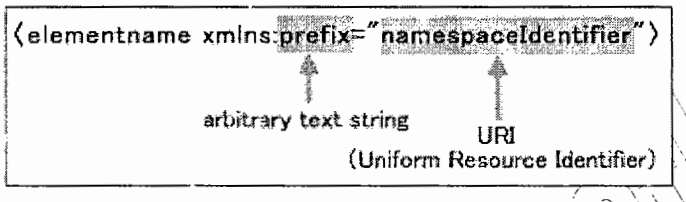
XML Namespace is a mechanism to avoid name conflicts by differentiating elements or attributes within an XML document that may have identical names, but different definitions.

The W3C recommended a specification called "Namespaces in XML," whereby XML vocabularies are mutually differentiated, allowing for the re-use of a vocabulary. Utilizing this XML namespace allows us to avoid any element name conflicts.

```
<student>
    <no>10</no>
    <sname>Ganapathi</sname>
</student>

<employee>
   <no>10</no>
   <ename>Raju</ename>
</employee>
```

We will encounter a problem if we try to combine the above documents. This is because they both have an element called "no". One is the "no" of the "student", the other is the "no" of the "employee" page. We have a name conflict.

Write a namespace declaration according to the following description method, describing the element start tag:



If the element and/ or attribute belong to a namespace, a colon (":") is placed between the namespace prefix and the element name/ attribute name.

```
<stu:student  xmlns:stu="http://www.rajuhelps2u.com/student/">
 <stu:no>10</stu:no>
 <stu:sname>Ganapthi<stu:sname>
</stu:student>
<emp:employee xmlns:emp="http://www.rajuhelps2u.com/employee/">
 <emp:no>10</emp:no>
 <emp:ename>Raju<emp:ename>
</emp:emploee>
```

We have added the xmlns:{prefix} attribute to the root element. We have assigned this attribute a unique value. This unique value is usually in the form of a Uniform Resource Identifier (URI). This defines the namespace.

And, now that the namespace has been defined, we have added a "stu" and "emp" prefix to our element names.

Now, when we combine the two documents, the XML processor will see two different element names: stu:no (from the Student document) and emp:no (from the Employee document).

Multiple Namespaces:

You could also have multiple namespaces within your XML document. For example, you could define one namespace against the root element, and another against a child element.

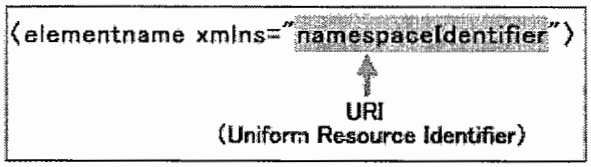
Example:

```
<college xmlns:stu="http://rajuhelps2u.com/student/"
         xmlns:emp="http://rajuhelps2u.com/employee/">
```

```
<stu:student>
 <stu:no>10</stu:no>
 <stu:name>Ganapthi</stu:name>
</stu:student>
<emp:employee>
 <emp:no>10</emp:no>
 <emp:name>Raju</emp:name>
</emp:emploee>
</college>
```

Default Namespaces:

A "default namespace" is a namespace declaration that does not use a namespace prefix as shown in below diagram. The scope of the default namespace is the element for which the namespace was declared and the related content, just as with the namespace scope discussed earlier. The benefit of using a default namespace is that the namespace prefix can be omitted.



Example:

```
<college xmlns="http://rajuhelps2u.com/student/"
         xmlns:emp="http://rajuhelps2u.com/employee/">
<student>
 <no>10</no>
 <name>Ganapthi</name>
</student>
<emp:employee>
 <emp:no>10</emp:no>
 <emp:name>Raju</emp:name>
</emp:emploee>
</college>
```

Declaring Name Space:

1. Declaring namespace in the XSD using targetNameSpace
2. Using the elements that are declared under namespace in the xml document.

XSD Document:

1.student.xsd

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified" attributeFormDefault="unqualified"
targetNamespace="http://www.rajuhelps2u.com/student"
xmlns:stu="http://www.rajuhelps2u.com/stuent">
```

From the above code we are concluding the below points.

1. xmlns:xs

   elements and data types come from the http://www.w3.org/2001/XMLSchema

2. targetNamespace

   user defined elements and data types in our schema belongs to our namespace such as targetNamespace=http://www.rajuhelps2u.com/student

3. xmlns – indicates default namespace
4. elementFormDefault –elements must be namespace qualified
5. attributeFormDefault – attributes must be namespace qualified.

XML Document:

2.student.xml

```
<stu:student xmlns:stu="http://www.rajuhelps2u.com/student"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.rajuhelps2u.com/student
file:///G:/Batch/XMLNamespace/student.xsd">
```

From the above code we are concluding the below points.

schemaLocation :it contains two parts:

1.name space from which you are using elements.
2.document which contains this namespace.

a)xsi:schemaLocation=http://www.rajuhelsp2u.com/training file:///G:/Batch/XML%20Namespace/student.xsd
b) xsi:noNamespaceSchemaLocation ="student.xsd"

2. XML Schema-instance tells the XML parser that this document should be validated against a schema. it allows you to define the location of your XML schema.

XML schema in multiple files:

1. Declare two xsd documents with suffix as .xsd
2. Refer one document with another with
   <xs:include schemaLocation ="incluedfile.uri">

XML schemas with multiple namespaces:

<xs:import namespace ="uri" schemaLocation ="schema.uri"/>

Practical Example:

1.student.xsd

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:stu="http://www.rajuhelps2u.com/student"
targetNamespace="http://www.rajuhelps2u.com/student" elementFormDefault="qualified"
attributeFormDefault="qualified">
      <xs:element name="student">
            <xs:complexType>
                  <xs:sequence>
                        <xs:element name="no" type="xs:integer"/>
                        <xs:element name="name" type="xs:string"/>
                        <xs:element name="age" type="xs:integer"/>
      </xs:sequence>
            </xs:complexType>
      </xs:element>
</xs:schema>
```

2.student.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<stu:student xmlns:stu="http://www.rajuhelps2u.com/student"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.rajuhelps2u.com/student
file:///E:/Batch/XML/Namespace/student.xsd">
      <stu:no>10</stu:no>
      <stu:name>Raju</stu:name>
      <stu:age>22</stu:age>
</stu:student>
```

JAX-P

The Java API for XML Processing (JAXP) is for processing XML data using applications written in the Java programming language.
It is a specifications from W3C and its an API from Sun.
JAX-P API is implemented by multiple vendors

1. Xerces
2. Crimson
3. Oracle

Jdk 1.5 onwards Xerces is default parser.
XML processing methodologies using JAX-P

1. DOM
2. SAX

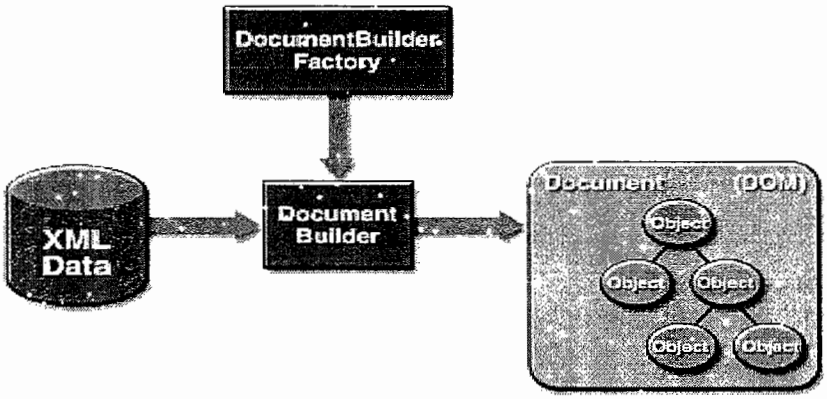
Overview of the main JAXP API Packages:

javax.xml.parsers	: The JAXP APIs provide a common interface for different vendors' to use SAX and DOM parsers.
org.w3c.dom	:Defines the Document class (a DOM) along with the classes for all of the components of a DOM.
org.xml.sax	:Defines the basic SAX APIs.
javax.xml.transform	:Defines the XSLT APIs that let's to transform XML into other forms

DOM

DOM stands for Document Object Model.
The DOM presents an XML document as a tree-structure.
The XML DOM defines a standard way for accessing and manipulating XML documents
The W3C Document Object Model (DOM) is a platform and language-neutral interface that allows programs and scripts to dynamically access and update the content, structure, and style of a document.
XML documents have a hierarchy of informational units called nodes; DOM is a way of describing those nodes and the relationships between them.
A DOM Document is a collection of nodes or pieces of information organized in a hierarchy. This hierarchy allows a developer to navigate through the tree looking for specific information. Because it is based on a hierarchy of information, the DOM is said to be tree based.
The XML DOM also provides an API that allows a developer to add, edit, move, or remove nodes in the tree at any point in order to create an application.
DOM is R/W parser
It immediate reads the entire XML into the memory.

Functional Diagram:



org.w3c.dom
Defines the DOM programming interfaces for XML (and, optionally, HTML) documents, as specified by the W3C.
javax.xml.parsers
Defines the DocumentBuilderFactory class and the DocumentBuilder class, which returns an object that implements the W3C Document interface. The factory that is used to create the builder is determined by the javax.xml.parsers system property, which can be set from the command line or overridden when invoking the new Instance method. This package also defines the ParserConfigurationException class for reporting errors

Import XML-related packages

You will need to import below packages first in your application.

```
import org.w3c.dom.*;
import javax.xml.parsers.*;
import java.io.*;
```

Create a DocumentBuilder

Next step is to create the DocumentBuilder object.

```
DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
```

DocumentBuilder builder = factory.newDocumentBuilder();

Create a Document from a file

Document document = builder.parse(new File(file));

Extract the root element

You can get the root element from XML document using below code.

Element root = document.getDocumentElement();

Examine attributes

You can examine the node attributes using below methods.

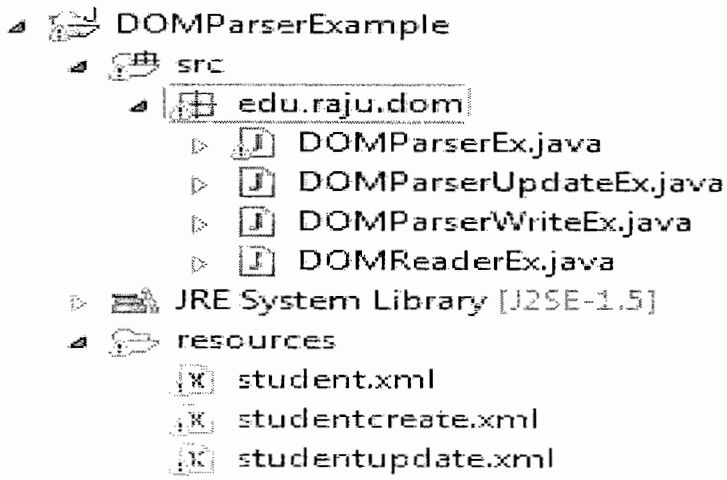
element.getAttribute("attributeName") ; //returns specific attribute
element.getAttributes(); //returns a Map (table) of names/values

Examine sub-elements:

Child elements can inquired in below manner.

node.getElementsByTagName("subElementName") //returns a list of sub-elements of specified name
node.getChildNodes() //returns a list of all child nodes

Examples:



1.student.xml

```
<!-- Student Data -->
<student>
      <sno>10</sno>
      <sname>Raju</sname>
      <sage>22</sage>
</student>
```

2.DomparserReaderEx.java

```
package edu.raju.dom;
```

```
import java.io.File;
import java.io.IOException;

import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.ParserConfigurationException;

import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.Node;
import org.w3c.dom.NodeList;
import org.xml.sax.SAXException;

public class DOMReaderEx {
      public static void main(String args[]) throws ParserConfigurationException,
                  SAXException, IOException {
            // Step -1: Create DocumentBuilderFactory
            DocumentBuilderFactory factory =
                  DocumentBuilderFactory.newInstance();
            factory.setIgnoringElementContentWhitespace(false);
            // Step -2: Create DocumentBuilder
            DocumentBuilder builder =
                  factory.newDocumentBuilder();


            // Step -3: Load XML document
            File file = new File(
                        "E:\\workspace\\Nit_WebService\\" +
                        "DOMParserExample\\resources\\student.xml");

            // Step -4:Parse XML document
            Document document = builder.parse(file);

            System.out.println("SUCCESS");

            Element rootElement =document.getDocumentElement();

            //get <sno> value
            Node first =rootElement.getFirstChild();
            Node finalNode =first.getFirstChild();
            String value =finalNode.getNodeValue();
            System.out.println(value);


            NodeList nodeList =rootElement.
            getElementsByTagName("sname");

            Node snameElement =nodeList.item(0);
            Node snameChild =snameElement.getFirstChild();
            String snamevalue  =snameChild.getNodeValue();
            System.out.println(snamevalue);

      }
}
```

3.DomparserWriterEx.java

```
package edu.raju.dom;

import java.io.File;
import javax.xml.parsers.DocumentBuilder;
```

```
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.ParserConfigurationException;
import javax.xml.transform.Transformer;
import javax.xml.transform.TransformerException;
import javax.xml.transform.TransformerFactory;
import javax.xml.transform.dom.DOMSource;
import javax.xml.transform.stream.StreamResult;

import org.w3c.dom.Attr;
import org.w3c.dom.Document;
import org.w3c.dom.Element;

public class DOMParserWriteEx {
      public static void main(String argv[]) {
            try {

                  DocumentBuilderFactory docFactory = DocumentBuilderFactory
                              .newInstance();
                  DocumentBuilder docBuilder = docFactory.newDocumentBuilder();

                  // root elements
                  Document doc = docBuilder.newDocument();
                  Element rootElement = doc.createElement("student");
                  doc.appendChild(rootElement);

                  // sno elements
                  Element snoElement = doc.createElement("sno");
                  snoElement.appendChild(doc.createTextNode("10"));
                  rootElement.appendChild(snoElement);

                  // set attribute to sno element
                  Attr attr = doc.createAttribute("id");
                  attr.setValue("1");
                  snoElement.setAttributeNode(attr);

                  // shorten way


                  // name elements
                  Element snameElement = doc.createElement("sname");
                  snameElement.appendChild(doc.createTextNode("Venu"));
                  rootElement.appendChild(snameElement);

                  // age elements
                  Element sageElement = doc.createElement("sage");
                  sageElement.appendChild(doc.createTextNode("32"));
                  rootElement.appendChild(sageElement);


                  // write the content into xml file
                  TransformerFactory transformerFactory = TransformerFactory
                              .newInstance();
                  Transformer transformer = transformerFactory.newTransformer();
                  DOMSource source = new DOMSource(doc);
                  StreamResult result = new StreamResult(new
File("G:\\Workspaces\\Batch\\DOMParserExample\\resources\\studentcreate.xml"));

                  // Output to console for testing
                  // StreamResult result = new StreamResult(System.out);
```

```
                transformer.transform(source, result);

                System.out.println("File saved!");

            } catch (ParserConfigurationException pce) {
                pce.printStackTrace();
            } catch (TransformerException tfe) {
                tfe.printStackTrace();
            }
        }

}
```

4.studentcreate.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<student>
      <sno id="1">10</sno>
      <sname>Venu</sname>
      <sage>32</sage>
</student>
```

5.DomparserUpdateEx.java

```
package edu.raju.dom;

import java.io.File;
import java.io.IOException;

import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.ParserConfigurationException;
import javax.xml.transform.Transformer;
import javax.xml.transform.TransformerException;
import javax.xml.transform.TransformerFactory;
import javax.xml.transform.dom.DOMSource;
import javax.xml.transform.stream.StreamResult;

import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.Node;
import org.w3c.dom.NodeList;
import org.xml.sax.SAXException;

public class DomparserUpdateEx {
      public static void main(String argv[]) throws SAXException, IOException {
            try {

                DocumentBuilderFactory docFactory = DocumentBuilderFactory
                            .newInstance();
                DocumentBuilder docBuilder = docFactory.newDocumentBuilder();

                File file = new File(

      "E:\\workspace\\Nit WebService\\DOMParserExample\\resources\\studentupdate.xml
");

                Document document = docBuilder.parse(file);
```

```
                System.out.println("SUCCESS");

                Element rootElement = document.getDocumentElement();

                NodeList list = rootElement.getChildNodes();

                for (int i = 0; i < list.getLength(); i++) {

                      Node node = list.item(i);

                      // get the sno element, and update the value
                      if ("sno".equals(node.getNodeName())) {
                            node.setTextContent("30");
                      }

                      // remove sname
                      if ("sname".equals(node.getNodeName())) {
                            rootElement.removeChild(node);
                      }

                }
                // write the content into xml file
                TransformerFactory transformerFactory = TransformerFactory
                            .newInstance();
                Transformer transformer = transformerFactory.newTransformer();
                DOMSource source = new DOMSource(document);
                StreamResult result = new StreamResult(
                            new File(

      "E:\\workspace\\Nit_WebService\\DOMParserExample\\resources\\studentupdate.xml
"));

                // Output to console for testing
                // StreamResult result = new StreamResult(System.out);

                transformer.transform(source, result);

                System.out.println("File saved!");

          } catch (ParserConfigurationException pce) {
                pce.printStackTrace();
          } catch (TransformerException tfe) {
                tfe.printStackTrace();
          }
      }

}
```

6.studentupdate.xml

```
<?xml version="1.0" encoding="UTF-8"?><student>
          <sno>30</sno>

          <sage>22</sage>
</student>
```

7.DomparserExample.java

```
package edu.raju.dom;

import java.io.File;
```

```
import java.io.IOException;

import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.ParserConfigurationException;

import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.Node;
import org.w3c.dom.NodeList;
import org.xml.sax.SAXException;

public class DomparserExample {
            public static void main(String args[]) throws
ParserConfigurationException,
                  SAXException, IOException {
            // Step -1: Create DocumentBuilderFactory
            DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
            factory.setIgnoringElementContentWhitespace(true);
            factory.setIgnoringComments(true);

            // Step -2: Create DocumentBuilder
            DocumentBuilder builder = factory.newDocumentBuilder();

            // Step -3: Load XML document
            File file = new File(

            "E:\\workspace\\Nit_WebService\\DOMParserExample\\resources\\student.xml"
);

            // Step -4:Parse XML document
            Document document = builder.parse(file);

            System.out.println("SUCCESS");
            echo(document);

            }

            public static void echo(Node node) {
            if (node != null) {
                  int type = node.getNodeType();
                  switch (type) {

                  case Node.DOCUMENT_NODE:
                        System.out.println("<?xml version ='1.0'?>");
                        echo(node.getFirstChild());
                        break;
                  case Node.ELEMENT_NODE:
                        System.out.println("<" + node.getNodeName() + ">");
                        NodeList childrens = node.getChildNodes();
                        if (childrens != null) {
                              for (int i = 0; i < childrens.getLength(); i++) {
                                    Node child = childrens.item(i);
                                    echo  (child);
                              }
                        }
                        System.out.println("</" + node.getNodeName() + ">");
                        break;
                  case Node.TEXT_NODE:
```

```
                    System.out.println(node.getNodeValue());
                    break;
                }

            }
            }
}
```

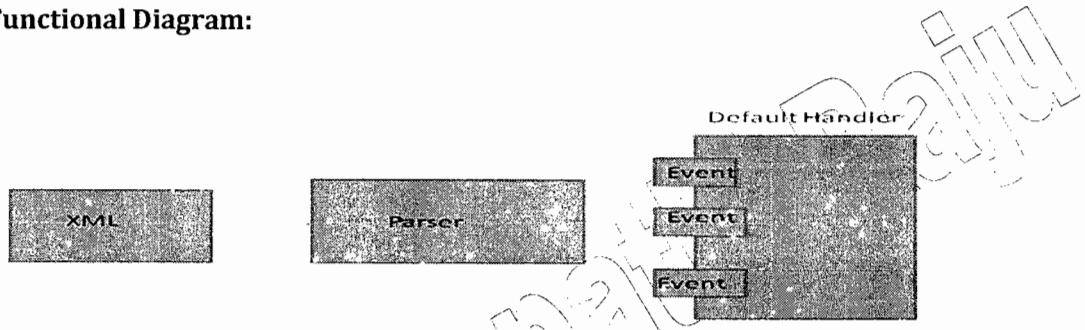
SAX

SAX stands for Simple Access for XML
It is a methodology that allows reading of XML documents in an event base processing model.
SAX Parser is different from the DOM Parser where SAX parser doesn't load the complete XML into the memory, instead it parses the XML line by line triggering different events as and when it encounters different elements like: opening tag, closing tag, character data, comments and so on. This is the reason why SAX Parser is called an event based parser.
SAX supports validation. We can check with DTD/Schema
SAX is fast and take less amount of memory.
Event Based Model:

1.Source: It is an originator of event.
event: button click, mouse over, key pressed.
2.Listner :It listens for an event from the source.
3.Event Handler: Once a listener captures an event, It calls a method on the event handler class.

In case of SAX the source is XML document
Handler is the class which containing methods to execute on behalf of the event.
Parser acts as a the listener who will listen for the event on source and triggers a method call on the handler.

Functional Diagram:



javax.xml.parsers.SAXParser:
it provides method to parse XML document using event handlers. This class implements XMLReader interface and provides overloaded versions of parse() methods to read XML document from File, InputStream, SAX InputSource and String URI.

The actual parsing is done by the Handler class and we need to create our own handler class to parse the XML document. We need to implement org.xml.sax.ContentHandler interface to create our own handler classes. This interface contains callback methods that receive notification when any event occurs, for example StartDocument, EndDocument, StartElement, EndElement, CharacterData etc.

org.xml.sax.helpers.DefaultHandler provides default implementation of ContentHandler interface and we can extend this class to create our own handler. It's advisable to extend this class because we might need only few of the methods to implement. Extending this class will keep our code cleaner and maintainable.

Default Handler Methods:

1.startDocument: It is triggered at the start of the XML document.
2.startElement: whenever the parser encounters the starting element. it raises this method call by passing the entire information to the method.
3. Characters : This method would be invoked by the parser. whenever it encounters data portion between the XML elements.

To this method it passes the entire XML as character array along with two other integers one indicating the position in the array from which the current data portion begins and the second integer representing the number of characters the data span to.

4.endElement: Triggered by the parser when it encounters a closing element.

5.endDocument: Triggered by the parser once it reaches end of the document

Examples:

1. student.xml

```
<student>
	<sno>10</sno>
	<sname>Raju</sname>
	<sage>22</sage>
</student>
```

2. student.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
	<xs:element name="student">
		<xs:complexType>
			<xs:sequence>
				<xs:element name="sno" type="xs:integer"/>
				<xs:element name="sname" type="xs:string"/>
				<xs:element name="sage" type="xs:string"/>
			</xs:sequence>
		</xs:complexType>
	</xs:element>
</xs:schema>
```

3. StudentHandler.java

```
package edu.raju.handler;

import org.xml.sax.Attributes;
import org.xml.sax.SAXException;
import org.xml.sax.helpers.DefaultHandler;
public class StudentHandler extends DefaultHandler {

	@Override
	public void startDocument() throws SAXException {
		System.out.println("START DOCUMENT");
	}

	@Override
	public void startElement(String arg0, String arg1, String localName,
			Attributes arg3) throws SAXException {

		System.out.print("<"+localName+">");
	}

	@Override
	public void characters(char[] xml, int offSet, int len) throws SAXException {
		String data = new String(xml, offSet, len);
		System.out.print(data);

	}

	@Override
	public void endElement(String arg0, String arg1, String localName)
			throws SAXException {
		System.out.println("</"+localName+">");
	}
	@Override
	public void endDocument() throws SAXException {
```

```
            System.out.println("END DOCUMENT");
      }
}
```

4.StudentParserEx.java

```
package edu.raju.saxparser.parser;

import java.io.File;
import java.io.IOException;

import javax.xml.XMLConstants;
import javax.xml.parsers.ParserConfigurationException;
import javax.xml.parsers.SAXParser;
import javax.xml.parsers.SAXParserFactory;
import javax.xml.transform.stream.StreamSource;
import javax.xml.validation.Schema;
import javax.xml.validation.SchemaFactory;
import javax.xml.validation.Validator;

import org.xml.sax.SAXException;

import edu.raju.handler.StudentHandler;

public class StudentParserEx {
      public static void main(String args[]) throws ParserConfigurationException,
                  SAXException, IOException {
            SchemaFactory sFactory = SchemaFactory
                        .newInstance(XMLConstants.W3C_XML_SCHEMA_NS_URI);
            Schema studentSchema = sFactory
                        .newSchema(new File(

      "G:\\Workspaces\\Batch\\SAXExample\\resources\\student.xsd"));
            Validator studentValidator = studentSchema.newValidator();
            studentValidator
                        .validate(new StreamSource(
                                    new File(

      "G:\\Workspaces\\Batch\\SAXExample\\resources\\student.xml")));


            System.out.println("It is Valid document");


            SAXParserFactory factory = SAXParserFactory.newInstance();
            SAXParser parser = factory.newSAXParser();

            parser.parse(
                        new File(

      "G:\\Workspaces\\Batch\\SAXExample\\resources\\student.xml"),
                        new StudentHandler());
      }
}
```

Difference Between DOM and SAX?

DOM	SAX
Document Object Model	Simple Access for XML API
It is hierarchical processing model.	It is an event based processing model.
It supports random access of elements.	It process xml sequentially one after the another from top to bottom.
Stores the entire XML document into memory before processing	Doesn't store the XML in memory
It is R/W parser.	It is Read only Parser.
It is Slow.	It is Fast.
It consumes more memory.	It consumes less memory
Ease of navigation.	Backward navigation is not possible
Preserves comments	doesn't preserve comments

How to choose between DOM and SAX Parsers?

DOM Parser is Ideally a good parser should be fast (time efficient),space efficient, rich in functionality and easy to use . But in reality, none of the main parsers have all these features at the same time. For example, a DOM Parser is rich in functionality (because it creates a DOM tree in memory and allows you to access any part of the document repeatedly and allows you to modify the DOM tree), but it is space inefficient when the document is huge, and it takes a little bit long to learn how to work with it.

A SAX Parser, however, is much more space efficient in case of big input document (because it creates no internal structure). What's more, it runs faster and is easier to learn than DOM Parser because its API is really simple.
But from the functionality point of view, it provides less functions which mean that the users themselves have to take care of more, such as creating their own data structures.

JAX-B

Java Architecture for XML Binding

It is an API used for Java Objects to XML Documents and vice versa.

Java Architecture for XML Binding (JAXB) provides a fast and convenient way to bind XML schemas and Java representations, making it easy for Java developers to incorporate XML data and processing functions in Java applications. As part of this process, JAXB provides methods for unmarshalling (reading) XML instance documents into Java content trees, and then marshalling (writing) Java content trees back into XML instance documents.

JAXB also provides a way to generate XML schema from Java objects

Few binding frameworks

1.Caster

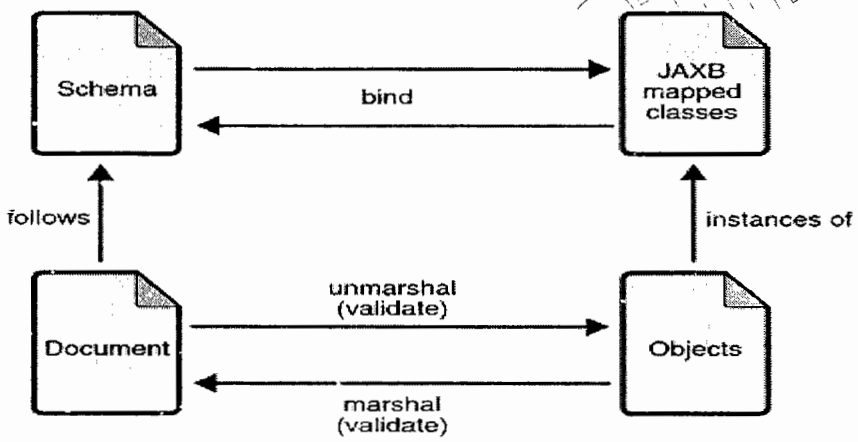
2.Glue

3.XML Beans,

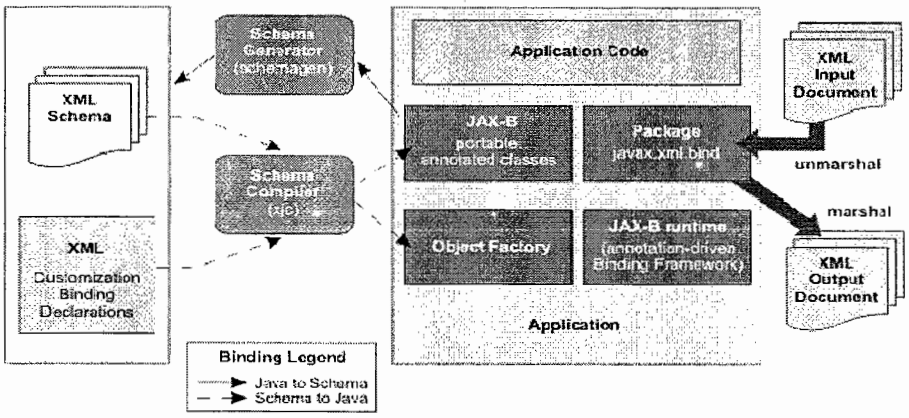
4.Xstream …

Jdk 1.6 onwards Jax-b is default.

Functional Diagrams of JAX-B:



Diagrametical Representation of JAX-B



Working with JAX- B:

1. Download:
   Download "JWSDP-2.0" (Java Webservice Developer Pack), which offers tools and libraries of (JAX-P,JAX-B,SAAJ,JAX-RPC and JAX-WS etc)
   Install JWSDP under c:\Sun\JWSDP-2.0
2. XJC: it is available in the below path:
   set path = c:\Sun\JWSDP-2.0\jaxb\bin
   it needs xsd as an input and generates as follows
   d:\project>xjc –d src resources\student.xsd
   1)Generate package based on the namespace:
   http://rajuhelsp2u.com/trainings/xml --> com.rajuhelsp2u.trainings.xml
   2)package-info.java àstores mapping information of namespace and package.
   3)Objectfactory.javaàits like factroy class.it has methods of creating an objects for binding classes. like createXXX .
   4)Binding classes: Binding classes are the classes which are bonded to XSD complexType declarations and denote this relationship,those are marked with jax-b annotations. i e @XMLTYPE ,@XMLELEMENT

Jax-B supports two types of operations

1. One-time operation:
   Which is done by only once ie. XSD to Binding classess and vice versa.
2. Runtime operation
   a)Marshalling : Converting java objects with data to its XML equivalent.
   b)Un-Marshalling : Converting XML to Java Object.
   c)In-Memory validation: Validate weather the XML confirms to XSD or not.

JAXB uses annotations to indicate the central elements.
Table 1.

Annotation	Description
@XmlRootElement(namespace = "namespace")	Define the root element for an XML tree
@XmlType(propOrder = { "field2", "field1",.. })	Allows to define the order in which the fields are written in the XML file

Annotation	Description
@XmlElement(name = "neuName")	Define the XML element which will be used. Only need to be used if the neuNeu is different then the JavaBeans Name

Why Jax-B?

1. JAXB parser is faster than SAX parser.
2. Its provides random access like DOM
3. Its provides compiler that compiles XML Schema to java classes

JAX-B Comparision with DOM/SAX:

1. JAXB is simple to use compared to SAX/DOM.
2. JAXB allows non-sequential processing of XML documents. In DOM we need to navigate a tree structure and JAXB doesn't requires such navigations.
3. We work with java instances instead of event/elements.
   JAXB uses memory efficiently than DOM.
4. We can unmarshal XML data .
5. Similarly we can marshal XMl content tree to different data targets.

Examples:

Step 1: write the following the xsd

1.student.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
      targetNamespace="http://rajuhelsp2u.com/trainings/xml"
xmlns:s="http://rajuhelsp2u.com/trainings/xml"
      elementFormDefault="qualified">
      <xs:element name="student" type="s:studentType"/>
      <xs:complexType name="studentType">
                  <xs:sequence>
                        <xs:element name="sno" type="xs:int" />
                        <xs:element name="sname" type="xs:string" />
                        <xs:element name="sage" type="xs:string" />
                        <xs:element name="address" type="s:addressType" />
                  </xs:sequence>

      </xs:complexType>

      <xs:complexType name="addressType">
            <xs:sequence>
                  <xs:element name="city" type="xs:string" />
                  <xs:element name="state" type="xs:string" />
                  <xs:element name="pin" type="xs:int" />
            </xs:sequence>
      </xs:complexType>

</xs:schema>
```

Step 2: Run the following command

- XJC: it is available in the below path:
- set path = c:\Sun\JWSDP-2.0\jaxb\bin
- d:\project>xjc –d src resources\student.xsd

- it needs xsd as an input and generates as follows
- its create a new package based on target namespace and place the following files.

2. StudentType.java

```
package com.rajuhelsp2u.trainings.xml;
import javax.xml.bind.annotation.XmlAccessType;
import javax.xml.bind.annotation.XmlAccessorType;
import javax.xml.bind.annotation.XmlElement;
import javax.xml.bind.annotation.XmlRootElement;
import javax.xml.bind.annotation.XmlType;
@XmlRootElement
@XmlAccessorType(XmlAccessType.FIELD)
@XmlType(name = "studentType", propOrder = {
    "sno",
    "sname",
    "sage",
    "address"
})
public class StudentType {
    protected int sno;
    @XmlElement(required = true)
    protected String sname;
    @XmlElement(required = true)
    protected String sage;
    @XmlElement(required = true)
    protected AddressType address;
    public int getSno() {
        return sno;
    }
    public void setSno(int value) {
        this.sno = value;
    }
    public String getSname() {
        return sname;
    }
    public void setSname(String value) {
        this.sname = value;
    }
    public String getSage() {
        return sage;
    }
    public void setSage(String value) {
        this.sage = value;
    }
    public AddressType getAddress() {
        return address;
    }
    public void setAddress(AddressType value) {
        this.address = value;
    }

}
```

3. AddressType.java

```
package com.rajuhelsp2u.trainings.xml;

import javax.xml.bind.annotation.XmlAccessType;
```

```
import javax.xml.bind.annotation.XmlAccessorType;
import javax.xml.bind.annotation.XmlElement;
import javax.xml.bind.annotation.XmlType;


@XmlAccessorType(XmlAccessType.FIELD)
@XmlType(name = "addressType", propOrder = {
    "city",
    "state",
    "pin"
})
public class AddressType {

    @XmlElement(required = true)
    protected String city;
    @XmlElement(required = true)
    protected String state;
    protected int pin;

    public String getCity() {
        return city;
    }

    public void setCity(String value) {
        this.city = value;
    }


    public String getState() {
        return state;
    }


    public void setState(String value) {
        this.state = value;
    }

    public int getPin() {
        return pin;
    }

    public void setPin(int value) {
        this.pin = value;
    }

}
```

4. package-Info.java

```
@javax.xml.bind.annotation.XmlSchema(namespace =
"http://rajuhelsp2u.com/trainings/xml", elementFormDefault =
javax.xml.bind.annotation.XmlNsForm.QUALIFIED)
package com.rajuhelsp2u.trainings.xml;
```

5.ObjectFactory.java

```
package com.rajuhelsp2u.trainings.xml;

import javax.xml.bind.JAXBElement;
import javax.xml.bind.annotation.XmlElementDecl;
```

```
import javax.xml.bind.annotation.XmlRegistry;
import javax.xml.namespace.QName;

ObjectFactory.java

@XmlRegistry
public class ObjectFactory {

    private final static QName _Student_QNAME = new
QName("http://rajuhelsp2u.com/trainings/xml", "student");

    public ObjectFactory() {
    }

    public AddressType createAddressType() {
        return new AddressType();
    }


    public StudentType createStudentType() {
        return new StudentType();
    }


    @XmlElementDecl(namespace = "http://rajuhelsp2u.com/trainings/xml", name =
"student")
    public JAXBElement<StudentType> createStudent(StudentType value) {
        return new JAXBElement<StudentType>(_Student_QNAME, StudentType.class, null,
value);
    }

}
```

Step 3: Write a test classes for marshalling and unmarshalling as given below.

6.StudentMarshaller.java

```
package edu.raju.test;

import javax.xml.bind.JAXBContext;
import javax.xml.bind.JAXBException;
import javax.xml.bind.Marshaller;

import com.rajuhelsp2u.trainings.xml.AddressType;
import com.rajuhelsp2u.trainings.xml.StudentType;

public class StudentMarshaller {
      public static void main(String args[]) throws JAXBException {
            AddressType addressType =new AddressType();
            addressType.setCity("Secuderabad");
            addressType.setState("A.P");
            addressType.setPin(500035);

            StudentType student =new StudentType();
            student.setSno(10);
            student.setSname("Raju");
            student.setSage("22");
            student.setAddress(addressType);
            JAXBContext jContext =
```

```
JAXBContext.newInstance("com.rajuhelsp2u.trainings.xml");
            Marshaller m = jContext.createMarshaller();
            m.marshal(student, System.out);
      }
}
```

7.StudentUnmarshaller.java

```
package edu.raju.test;
import java.io.File;
import javax.xml.XMLConstants;
import javax.xml.bind.JAXBContext;
import javax.xml.bind.JAXBElement;
import javax.xml.bind.JAXBException;
import javax.xml.bind.Unmarshaller;
import javax.xml.validation.Schema;
import javax.xml.validation.SchemaFactory;
import org.xml.sax.SAXException;
import com.rajuhelsp2u.trainings.xml.StudentType;
public class StudentUnmarshaller {
      public static void main(String args[]) throws JAXBException, SAXException {
            SchemaFactory sFactory = SchemaFactory
                        .newInstance(XMLConstants.W3C_XML_SCHEMA_NS_URI);
            Schema studentSchema = sFactory.newSchema(new File(

      "G:\\Workspaces\\Batch\\Student_JAXB\\resources\\student.xsd"));

            JAXBContext jContext =
JAXBContext.newInstance("com.rajuhelsp2u.trainings.xml");
            Unmarshaller un = jContext.createUnmarshaller();
            // set schema to validate
            un.setSchema(studentSchema);

            JAXBElement<StudentType> jElement = (JAXBElement<StudentType>) un
                        .unmarshal(new File(

      "G:\\Workspaces\\Batch\\Student_JAXB\\resources\\student.xml"));
            StudentType student = jElement.getValue();
            System.out.println(student.getAddress().getCity());

      }
}
```

What is Distributed Application?

- Software that executes on two or more computers in a network.
- A Distributed Application in is an application that is comprised of multiple objects. These objects may be defined by different management packs and may be managed on the same agent or on different agents.
- A distributed system consists of multiple autonomous computers that communicate through a computer network. The computers interact with each other in order to achieve a common goal
- A technology is said to be distributed if it's business objects are geographically dispersed (Across multiple JVM'S) and still communicating one another

Why distributed technology?

- Increased productivity.
- More effective
- Cost Savings
- Load balancing
- High processing
- Flexibility

Distributed Technologies.

1. Socket Programming
2. CORBA(Common Object Request Broker Architecture)
3. RMI(Remote Method Invocation)
4. EJB(Enterprise Java Beans)
5. WebSevices
6. Others.....

1.Socket Programming
Basic fundamental techniques to make use of socket programming.
Disadvantage :

1. Proxy and helper are implemented as part of our application.
2. Future extensions required proxy and helper changes.

2.CORBA(Common Object Request Broker Architecture)
It uses
IDL(interface definition language)
MOM(Message oriented middleware)
Ad: 1.Platform independent
Dis:1.Heavy weight,
2.Lanquage dependent

3.RMI(Remote Method Invocation)
Parts:
1.RMI Interface
2.RMI Registry
3.RMI Protocol
4.JNDI
A)Marshalling
B)UnMarshalling
Ad: 1.OpenSource 2.Platform independent
Dis: 1.Language dependant.
Steps
Start RMI Registry

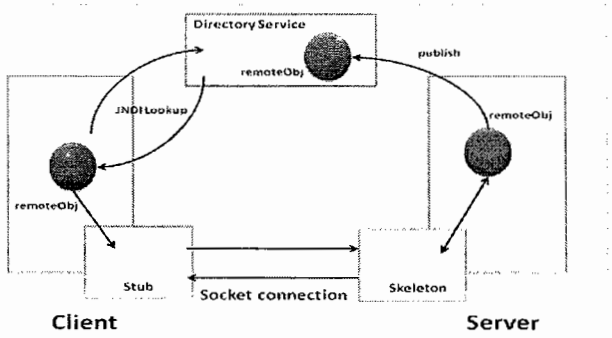
Compile Service Classes
RMIC tool

Stub:

1. Remote object address identifier
2. Binding parameters to be marshaled
3. Operation number describing method.

Skeleton:

1. Unmarshall the parameter
2. Call the desired method on real object.
3. Capture the return value
4. Marshall this value
5. Send package to marshaled form back to stub.



4.EJB(Enterprise Java Beans)

Enterprise application:

Application owned by organization to automate the business process

Product,Sales,Marketing,Accounts,Service

1.Security: Different departments needs to share information confidentially

2.Accuracy : o/p to i/p another departments

3.RMI :Different modules are deployed in different servers.

4.JPA : Info pass from one to another module.

Parts:

Enterprise application

Session Beans

Entity Beans

Message Driven Beans

Ad: 1.Managed objects

2.Container is providing all external dependencies

Dis: Specific to java

5.WebSevices

Ad:

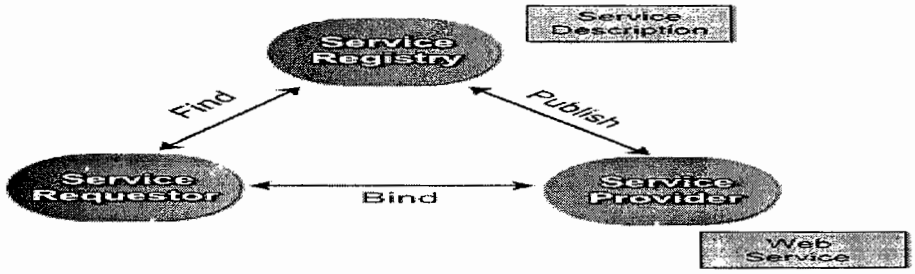
1. Platform independent
2. Language independent
3. Hardware and software independent
4. Interoperability
5. Loosely coupled

Web Service

Web Service

Web services extend the World Wide Web infrastructure to provide the means for software to connect to other software applications..
A **Web service** is a method of communication between two electronic devices over the world wide web.
A web service is a software designed to support interoperable machine to machine interaction over a network.

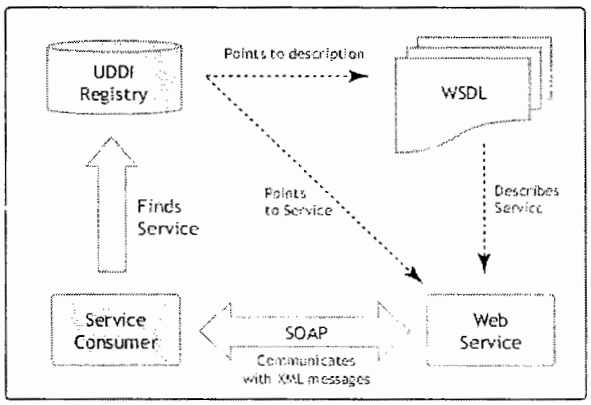
Basic Diagram of SOA:



Web Service Roles

There are three major roles within the web service architecture:
Service Provider: This is the provider of the web service. The service provider implements the service and makes it available on the Internet.
Service Requestor: This is any consumer of the web service. The requestor utilizes an existing web service by opening a network connection and sending an XML request.
Service Registry: This is a logically centralized directory of services. The registry provides a central place where developers can publish new services or find existing ones. It therefore serves as a centralized clearinghouse for companies and their services.

Web Service Functional Diagram:



SOAP:

- Simple Object Access Protocol
- SOAP is clarification protocol.
- SOAP is a binding protocol (Processing Data + Business Data)
- SOAP will be developed as a W3C standard.
- SOAP is a format for sending messages
- SOAP is designed to communicate via Internet
- SOAP is platform independent and language independent.
- Web Service Description Language.
- WSDL is pronounced as 'wiz-dull' and spelled out as 'W-S-D-L'
- WSDL is an XML document and Interoperable
- Which explains the services information.
- WSDL is the standard format for describing a web service.
- WSDL definition describes how to access a web service and what operations it will perform.

UDDI:

- Universal Description Discovery and Integration.
- It is a Registry where all the WSDL documents are registered.
- UDDI should be interoperable, It is also developed in XML
- It is a specification for a distributed registry of Web services.
- UDDI is platform independent, open framework
- UDDI can communicate via SOAP, CORBA, Java RMI Protocol.

Why Web service?

- Exposing the existing function on to network:
  Web Services allows you to expose the functionality of your existing code over the network..
- **Interoperability:**
  Web Services allows different applications to talk to each other and share data and services among themselves.
- **Standardized Protocol:**
  Web Services uses standardized industry standard protocol for the communication. All the four layers (Service Transport, XML Messaging, Service Description and Service Discovery layers) uses the well defined protocol in the Web Services protocol stack.
- **Low Cost of communication:**
  Web Services uses SOAP over HTTP protocol for the communication, so you can use your existing low cost internet for implementing Web Services.
- **XML-based**
  Web Services uses XML at data representation and data transportation layers.
- **Loosely coupled.**
- **Coarse-grained**
- **Ability to be synchronous or asynchronous:**
  Synchronicity refers to the binding of the client to the execution of the service. In synchronous invocations, the client blocks and waits for the service to complete its operation before continuing. Asynchronous operations allow a client to invoke a service and then execute other functions.
- **Supports Remote Procedure Calls (RPCs)**
  Web services allow clients to invoke procedures, functions, and methods on remote objects using an XML-based protocol.
- **Supports document exchange:**

WS-I (Web services Interoperability):

WS-I organization is an association of IT industry companies, including IBM and Microsoft, that aim to create web services specifications that all companies can use.

Interoperability: is the ability of diverse systems and organizations to work together (inter-operate).
Interoperability: is a property of a product or system, whose interfaces are completely understood, to work with other products or systems, present or future, without any restricted access or implementation.

WS-I standards?

1.Basic Profile 1.0(BP 1.0) --JAX-RPC
2.Basic Profile 2.0(BP 1.1) --JAX-WS

Web service

API	Implementations	Description
JAX-RPC	JAX- RPC	Java api for XML Remote procedure call sun implmentation
	Apache AXIS	Apache group
	Oracle Web logic servers	Oracle corp
	IBM Web Sphere web services	IBM
JAX-WS	JAX-WS	Reference Implementations.
	Metro	One more SUN implementations.
	Apache Axis2	Another apache group.
	Apache CXF	Apache group
	Oracle web logic web services	Oracle
	IBM web sphere web services	IBM

Web service development Parts.

1) JAX –RPC or JAX –WS
2) Two types of building web services.
    a. Contract first(top –down) approach (WSDL → services
    b. Contract last(bottom –up) approach (services →WSDL)
3) End point: A component which receives "Consumer" request.
    a. Servlet end point.
    b. Ejb end point.
4) Message Exchange Patterns (MEP):
    a. Synchronous.
    b. Asynchronous.
    c. Fire and Forget.
5) Message Exchange Formats (MEF):
    a. document-literal
    b. rpc-literal
    c. rpc-encoded.

JAX-RPC

JAX-RPC

JAX-RPC (Java API for XML-Based RPC) is an application program interface (API) in the Java Web Services Developer Pack (WSDP) that enables Java developers to include remote procedure calls (RPCs) with Web services or other Web-based applications.

JAX-RPC is aimed at making it easier for Web services to call other Web services.

JAX- API is released by Sun Microsystems

Its facilitate WS-I BP1.0 based web services.

It is intended to be a Java API to expose remote procedure calls that use XML to business applications that occur primarily.

JAX-RPC doesn't support Asynchronous Request pattern.

The default encoding technical is RPC-encoded.

JAX-RPC is a technology for building web services and clients that use remote procedure calls (RPC) and XML. Often used in a distributed client-server model, an RPC mechanism enables clients to execute procedures on other systems.

In JAX-RPC, a remote procedure call is represented by an XML-based protocol such as SOAP. The SOAP specification defines the envelope structure, encoding rules, and conventions for representing remote procedure calls and responses. These calls and responses are transmitted as SOAP messages (XML files) over HTTP.

Although SOAP messages are complex, the JAX-RPC API hides this complexity from the application developer. On the server side, the developer specifies the remote procedures by defining methods in an interface written in the Java programming language. The developer also codes one or more classes that implement those methods. Client programs are also easy to code. A client creates a proxy (a local object representing the service) and then simply invokes methods on the proxy. With JAX-RPC, the developer does not generate or parse SOAP messages. It is the JAX-RPC runtime system that converts the API calls and responses to and from SOAP messages.

Advantage:

- The platform independence of the Java programming language.
- JAX-RPC is not restrictive: a JAX-RPC client can access a web service that is not running on the Java platform, and vice versa.
- This flexibility is possible because JAX-RPC uses technologies defined by the World Wide Web Consortium (W3C): HTTP, SOAP, and the Web Service Description Language (WSDL). WSDL specifies an XML format for describing a service as a set of endpoints operating on messages.

SEI (Service End Point Interface):

The service endpoint interface defines the methods for particular Web services.
The JavaBeans implementation must implement methods with the same signature as the methods on the service endpoint interface.
A number of restrictions apply on which types to use as parameters and results of service endpoint interface methods. These restrictions are documented in the Java API for XML-based remote procedure call (JAX-RPC) specification, which are listed as following

Specifications:

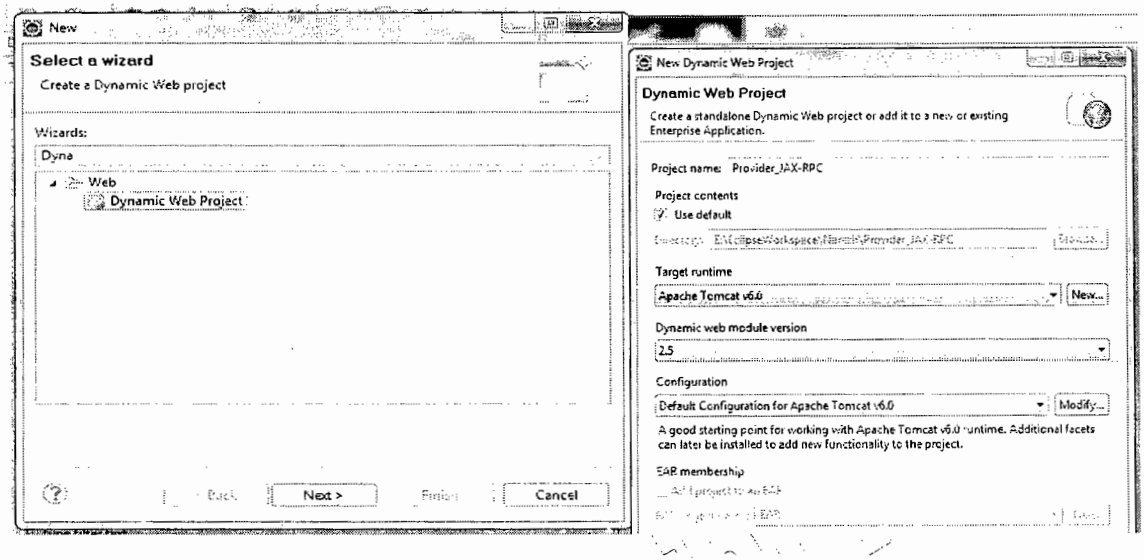
- Interface must extend java.rmi.Remote Interface.
- Declare the methods that you want to expose as web service methods.
- The methods parameters and return values should be serializable.
- Methods must throw Remote Exception.
- All the methods that declared in SEI interface must be Public.

Diagram:

JAX-RPC - Axis Implementation (Using Eclipse Galileo):

PROVIDER:

Step 1: Create a Dynamic Web Project (Provider_JAXPRC)



Step 2: Write an SEI interface and Implementation class.

IStudent.java

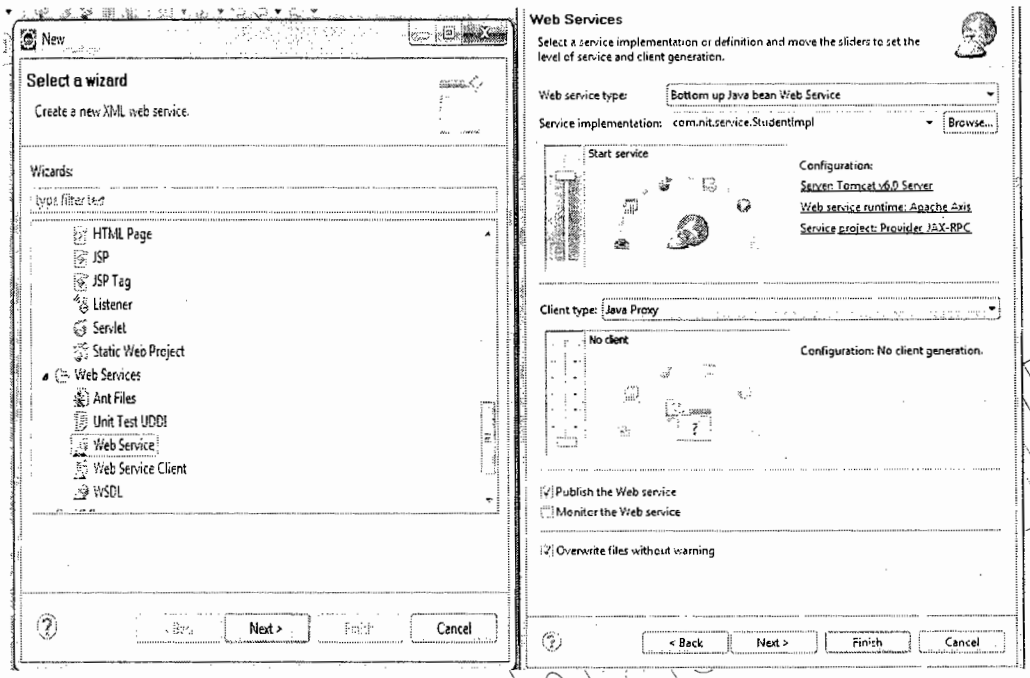
```
package com.nit.service;

import java.rmi.Remote;

import java.rmi.RemoteException;

public interface IStudent extends Remote{

        public String getStudentDetails(Integer sno)throws RemoteException;

}
```

StudentImpl.java

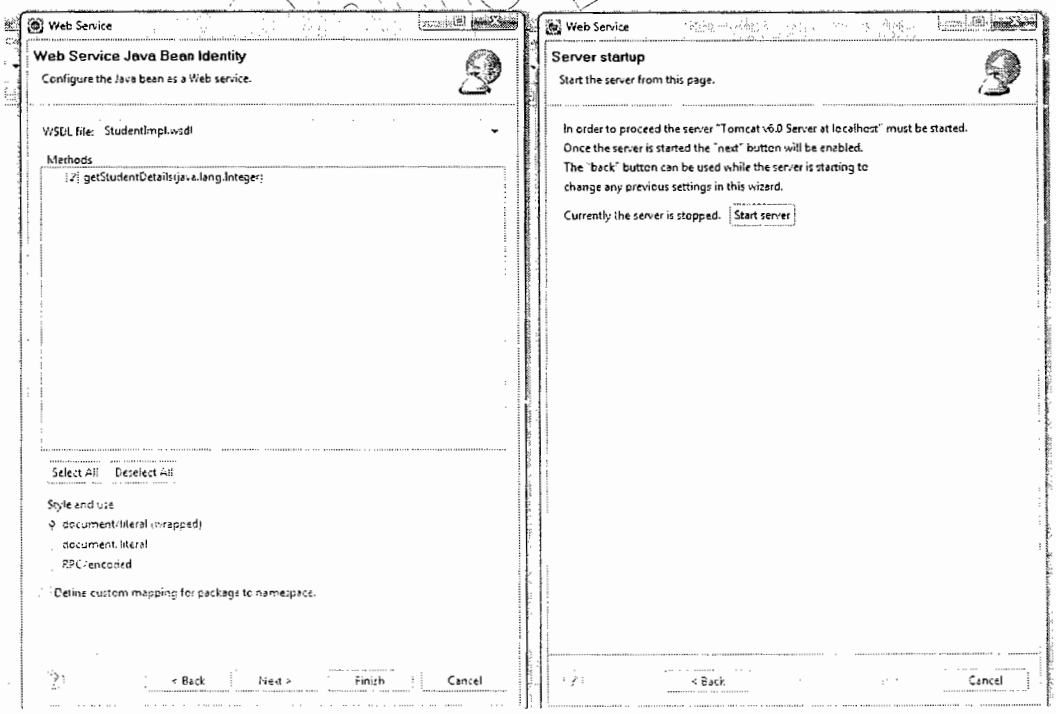
```
package com.nit.service;

import java.rmi.RemoteException;

public class StudentImpl implements IStudent {

        public String getStudentDetails(Integer sno) throws RemoteException {

                String name =null;

                if(sno == 001){

                    name ="Raju";

                }else if(sno == 002)

                {   name ="Rani";

                }else {

                     name ="Mantri";

                }
```

```
        return name;
    }
}
```

Step 2: Right click on project -->New-->Other-->Web Service
Step 3: Browse an Implementation class and select Bottom up java bean Web Service as shown below.
Step 4: Click on check box, Publish the Web Service

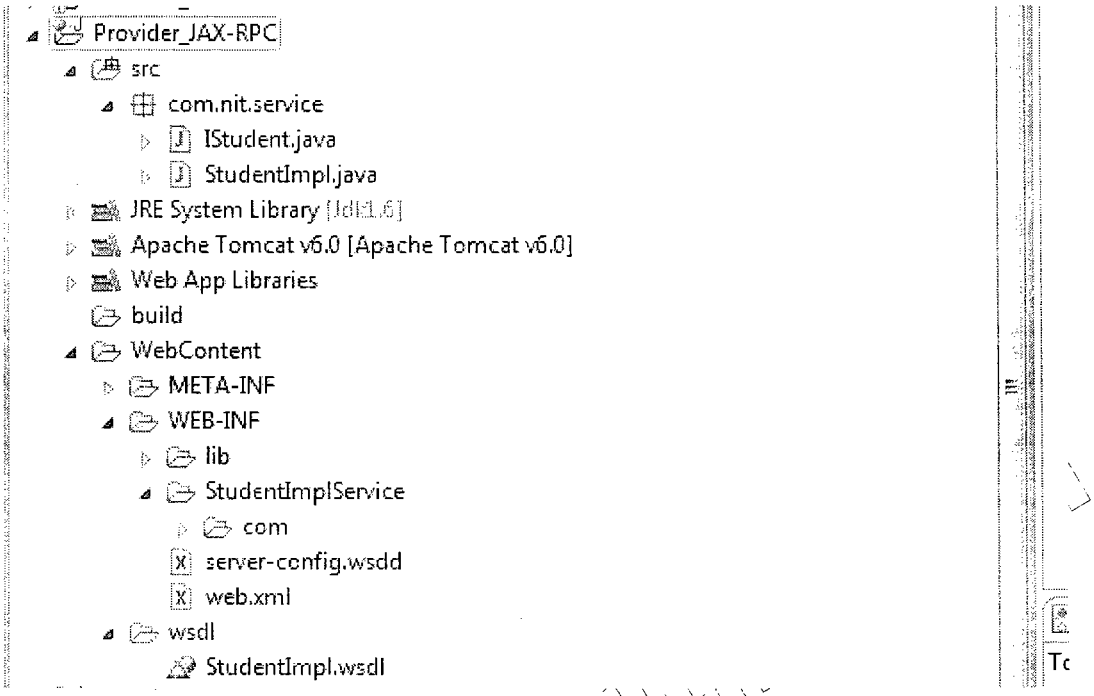


Step 5: Select the method which exposes as service.
Step 6: Start Server



Step 7: Observe the following.

a) WSDL file in WSDL folder
b) server-config.wsdd
c) web.xml



web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://java.sun.com/xml/ns/javaee"
xmlns:web="http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd" xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd" id="WebApp_ID" version="2.5">
 <display-name>Provider_JAX-RPC</display-name>
  <servlet>
   <display-name>Apache-Axis Servlet</display-name>
   <servlet-name>AxisServlet</servlet-name>
   <servlet-class>org.apache.axis.transport.http.AxisServlet</servlet-class>
 </servlet>
 <servlet-mapping>
   <servlet-name>AxisServlet</servlet-name>
   <url-pattern>/servlet/AxisServlet</url-pattern>
 </servlet-mapping>
 <servlet-mapping>
   <servlet-name>AxisServlet</servlet-name>
   <url-pattern>*.jws</url-pattern>
 </servlet-mapping>
 <servlet-mapping>
   <servlet-name>AxisServlet</servlet-name>
   <url-pattern>/services/*</url-pattern>
 </servlet-mapping>
 <servlet>
   <display-name>Axis Admin Servlet</display-name>
   <servlet-name>AdminServlet</servlet-name>
   <servlet-class>org.apache.axis.transport.http.AdminServlet</servlet-class>
   <load-on-startup>100</load-on-startup>
```

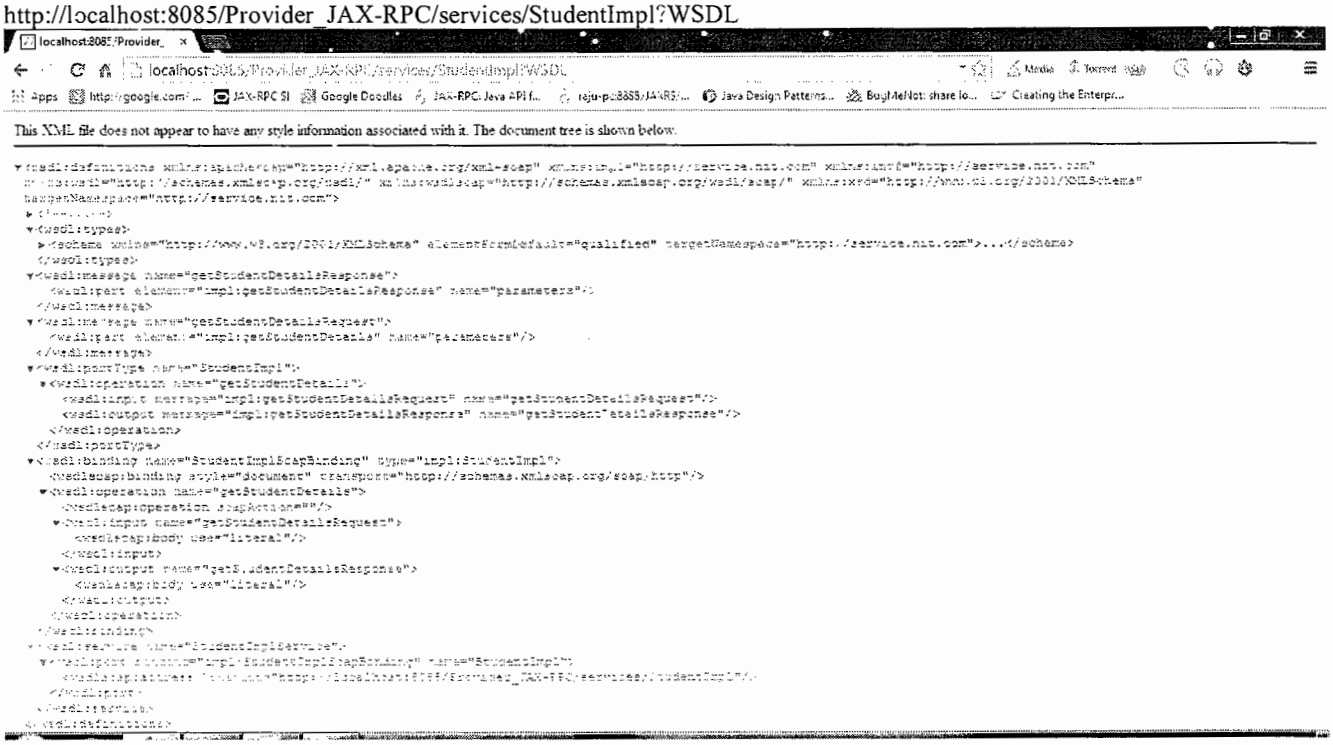
```
 </servlet>
 <servlet-mapping>
  <servlet-name>AdminServlet</servlet-name>
  <url-pattern>/servlet/AdminServlet</url-pattern>
 </servlet-mapping>
</web-app>
```

server-config.wsdd

```
<ns1:deployment xmlns="http://xml.apache.org/axis/wsdd/" xmlns:java="http://xml.apache.org/axis/wsdd/providers/java"
xmlns:ns1="http://xml.apache.org/axis/wsdd/">
 <ns1:globalConfiguration>
  <ns1:parameter name="sendMultiRefs" value="true"/>
  <ns1:parameter name="disablePrettyXML" value="true"/>
  <ns1:parameter name="adminPassword" value="admin"/>
  <ns1:parameter name="attachments.Directory"
value="E:\EclipseWorkspace\Naresh\.metadata\.plugins\org.eclipse.wst.server.core\tmp3\wtpwebapps\Provider_JAX-
RPC\WEB-INF\attachments"/>
  <ns1:parameter name="dotNetSoapEncFix" value="true"/>
  <ns1:parameter name="enableNamespacePrefixOptimization" value="false"/>
  <ns1:parameter name="sendXMLDeclaration" value="true"/>
  <ns1:parameter name="sendXsiTypes" value="true"/>
  <ns1:parameter name="attachments.implementation" value="org.apache.axis.attachments.AttachmentsImpl"/>
  <ns1:requestFlow>
   <ns1:handler type="java:org.apache.axis.handlers.JWSHandler">
    <ns1:parameter name="scope" value="session"/>
   </ns1:handler>
   <ns1:handler type="java:org.apache.axis.handlers.JWSHandler">
    <ns1:parameter name="scope" value="request"/>
    <ns1:parameter name="extension" value=".jwr"/>
   </ns1:handler>
  </ns1:requestFlow>
 </ns1:globalConfiguration>
 <ns1:handler name="URLMapper" type="java:org.apache.axis.handlers.http.URLMapper"/>
 <ns1:handler name="LocalResponder" type="java:org.apache.axis.transport.local.LocalResponder"/>
 <ns1:handler name="Authenticate" type="java:org.apache.axis.handlers.SimpleAuthenticationHandler"/>
 <ns1:service name="StudentImpl" provider="java:RPC" style="wrapped" use="literal">
  <ns2:operation name="getStudentDetails" qname="ns1:getStudentDetails" returnQName="ns1:getStudentDetailsReturn"
returnType="xsd:string" soapAction="" xmlns:ns1="http://service.nit.com"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:ns2="http://xml.apache.org/axis/wsdd/">
   <ns2:parameter qname="ns1:sno" type="xsd:int"/>
  </ns2:operation>
  <ns1:parameter name="allowedMethods" value="getStudentDetails"/>
  <ns1:parameter name="typeMappingVersion" value="1.2"/>
  <ns1:parameter name="wsdlPortType" value="StudentImpl"/>
  <ns1:parameter name="className" value="com.nit.service.StudentImpl"/>
  <ns1:parameter name="wsdlServicePort" value="StudentImpl"/>
  <ns1:parameter name="schemaQualified" value="http://service.nit.com"/>
  <ns1:parameter name="wsdlTargetNamespace" value="http://service.nit.com"/>
  <ns1:parameter name="wsdlServiceElement" value="StudentImplService"/>
 </ns1:service>
 <ns1:service name="AdminService" provider="java:MSG">
  <ns1:parameter name="allowedMethods" value="AdminService"/>
  <ns1:parameter name="enableRemoteAdmin" value="false"/>
  <ns1:parameter name="className" value="org.apache.axis.utils.Admin"/>
  <ns1:namespace>http://xml.apache.org/axis/wsdd/</ns1:namespace>
 </ns1:service>
 <ns1:service name="Version" provider="java:RPC">
  <ns1:parameter name="allowedMethods" value="getVersion"/>
```

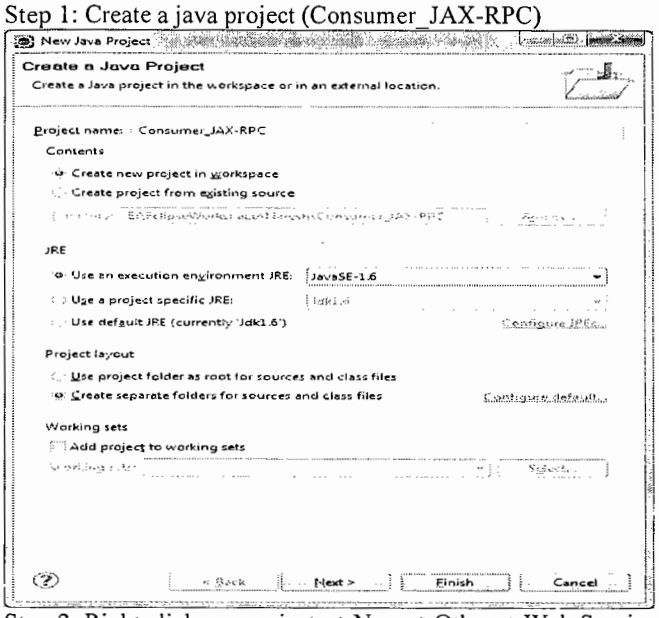
```
 <ns1:parameter name="className" value="org.apache.axis.Version"/>
</ns1:service>
<ns1:transport name="http">
 <ns1:requestFlow>
  <ns1:handler type="URLMapper"/>
  <ns1:handler type="java:org.apache.axis.handlers.http.HTTPAuthHandler"/>
 </ns1:requestFlow>
 <ns1:parameter name="qs:list" value="org.apache.axis.transport.http.QSListHandler"/>
 <ns1:parameter name="qs:wsdl" value="org.apache.axis.transport.http.QSWSDLHandler"/>
 <ns1:parameter name="qs.list" value="org.apache.axis.transport.http.QSListHandler"/>
 <ns1:parameter name="qs.method" value="org.apache.axis.transport.http.QSMethodHandler"/>
 <ns1:parameter name="qs:method" value="org.apache.axis.transport.http.QSMethodHandler"/>
 <ns1:parameter name="qs.wsdl" value="org.apache.axis.transport.http.QSWSDLHandler"/>
</ns1:transport>
<ns1:transport name="local">
 <ns1:responseFlow>
  <ns1:handler type="LocalResponder"/>
 </ns1:responseFlow>
</ns1:transport>
</ns1:deployment>
```

Step 8: Browse the following URL.
http://localhost:8085/Provider_JAX-RPC/services/StudentImpl?WSDL

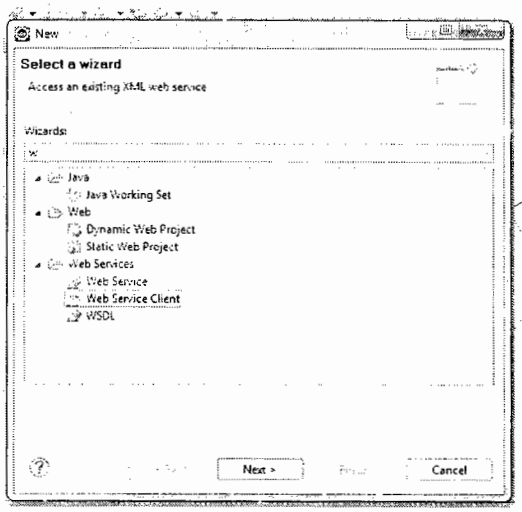


CONSUMER:

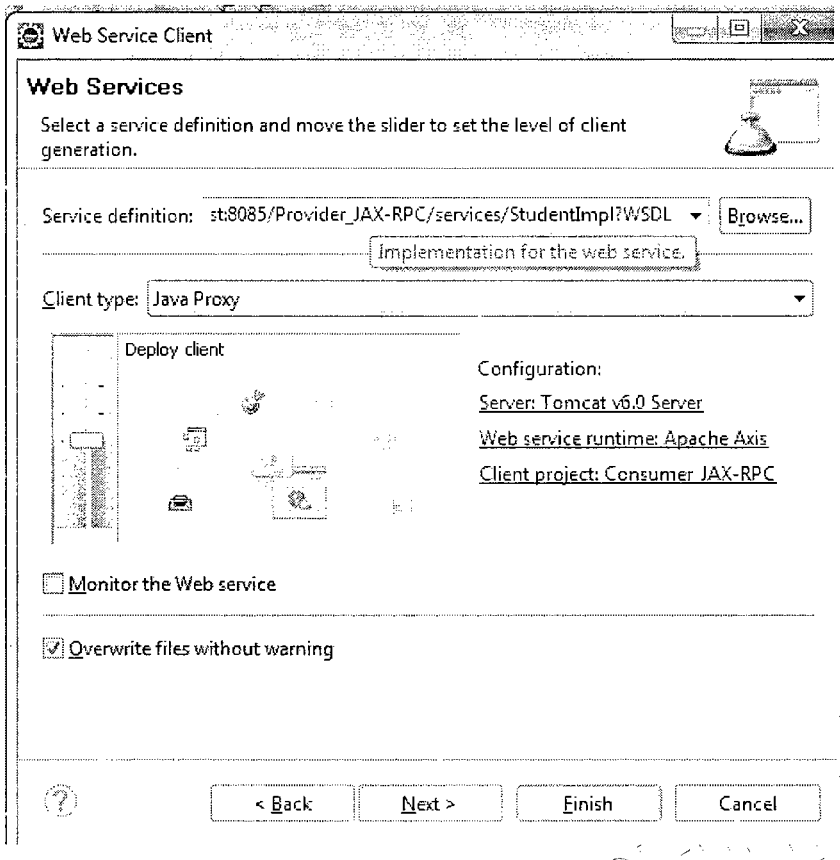
Step 1: Create a java project (Consumer_JAX-RPC)



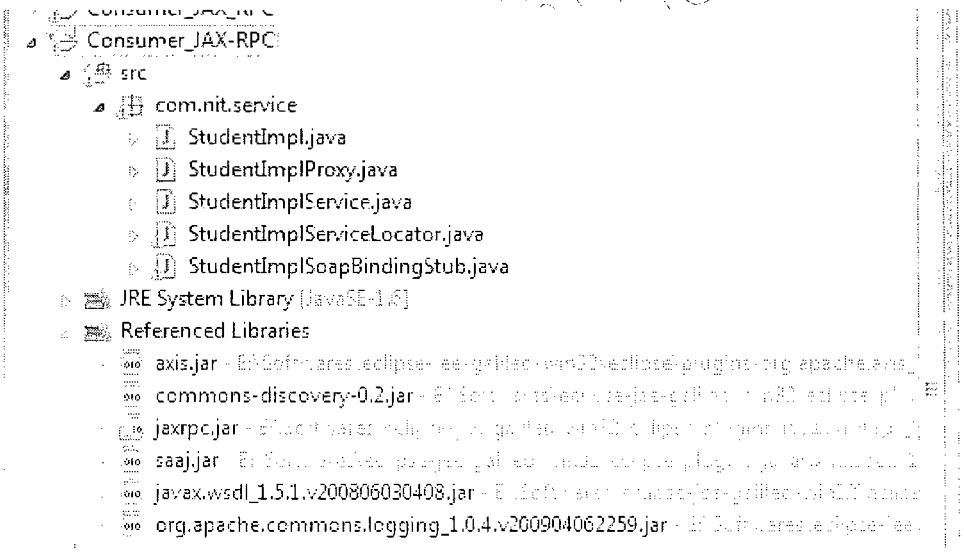
Step 2: Right click -on project -->New-->Other-->Web Service Client.



Step 3: Browse the URL of WSDL file as shown below screen.



Step 4: Observe the client-side artifacts in project.



Step 5: write an client class to test the application.

Consumer.java

```
package com.nit.test;
import java.rmi.RemoteException;
import javax.xml.rpc.ServiceException;
import com.nit.service.StudentImpl;
import com.nit.service.StudentImplServiceLocator;
```

```
public class Consumer {
        public static void main(String[] args) throws RemoteException, ServiceException {
                StudentImplServiceLocator serviceLocator =new StudentImplServiceLocator();
                StudentImpl inf =serviceLocator.getStudentImpl();
                String name =inf.getStudentDetails(001);
                System.out.println("Result:"+name);
        }
}
```

Testing Tools:

The following tools are used for testing of Web Services.

1. SOAP UI Tool.
2. Altova XML Spy:
3. Launch SOAP Web Services Explorer (My Eclipse IDE)
4. TCP/IP Monitor.

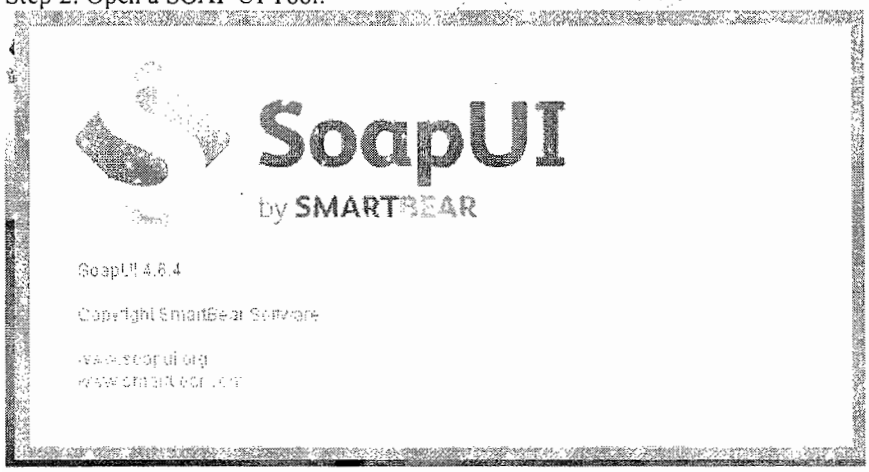
1.SOAP UI Tool

SoapUI is a free and open source cross-platform Functional Testing solution. With an easy-to-use graphical interface, and enterprise-class features, SoapUI allows you to easily and rapidly create and execute automated functional, regression, compliance, and load tests. In a single test environment, SoapUI provides complete test coverage and supports all the standard protocols and technologies. There are simply no limits to what you can do with your tests. Meet SoapUI, the world's most complete testing tool!

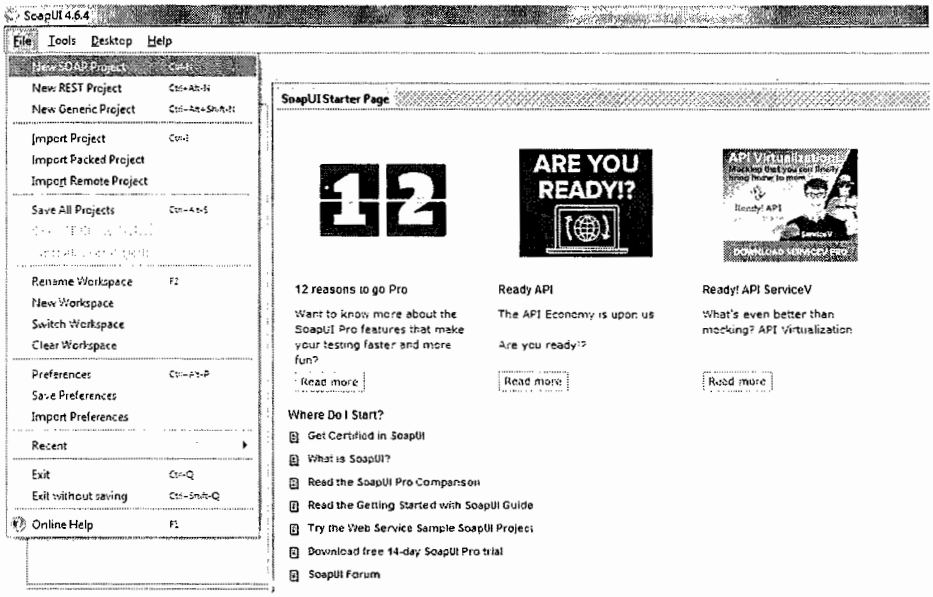
Even if you've never used SoapUI before, you'll find that creating even the most advanced test scenarios is very simple. Your testing journey in SoapUI begins with a Project. And creating one is as easy as a right-click. If you want, you can directly add a WSDL, create sample requests for all the operations in the service, and even create a mock of the imported WSDL – all at once.
You can even add a REST service – just click a checkbox and SoapUI takes care of the rest for you. Once you've created a project, instantly create and run any number of Functional/Load Tests, MockServices, and more. Thanks to the Navigator – the tree structure on the left of the main window – your progress and test organization are always in view. And from the Project window, you can manage and control everything related to your project.

Steps to Test Web Services Using SOAP UI Tool.

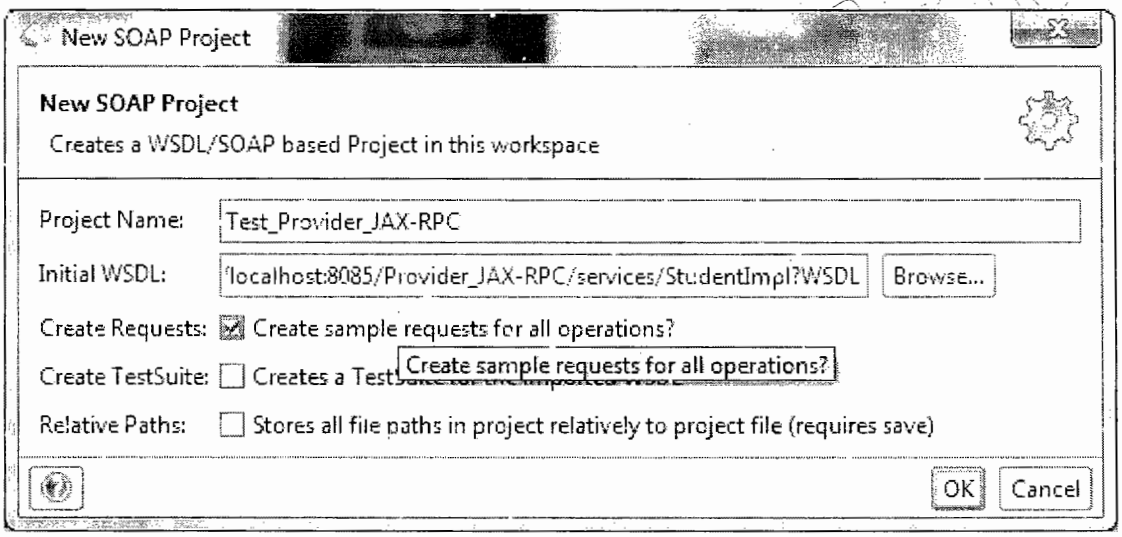
Step 1: Download and Install SOAP UI tool.
Step 2: Open a SOAP UI Tool.



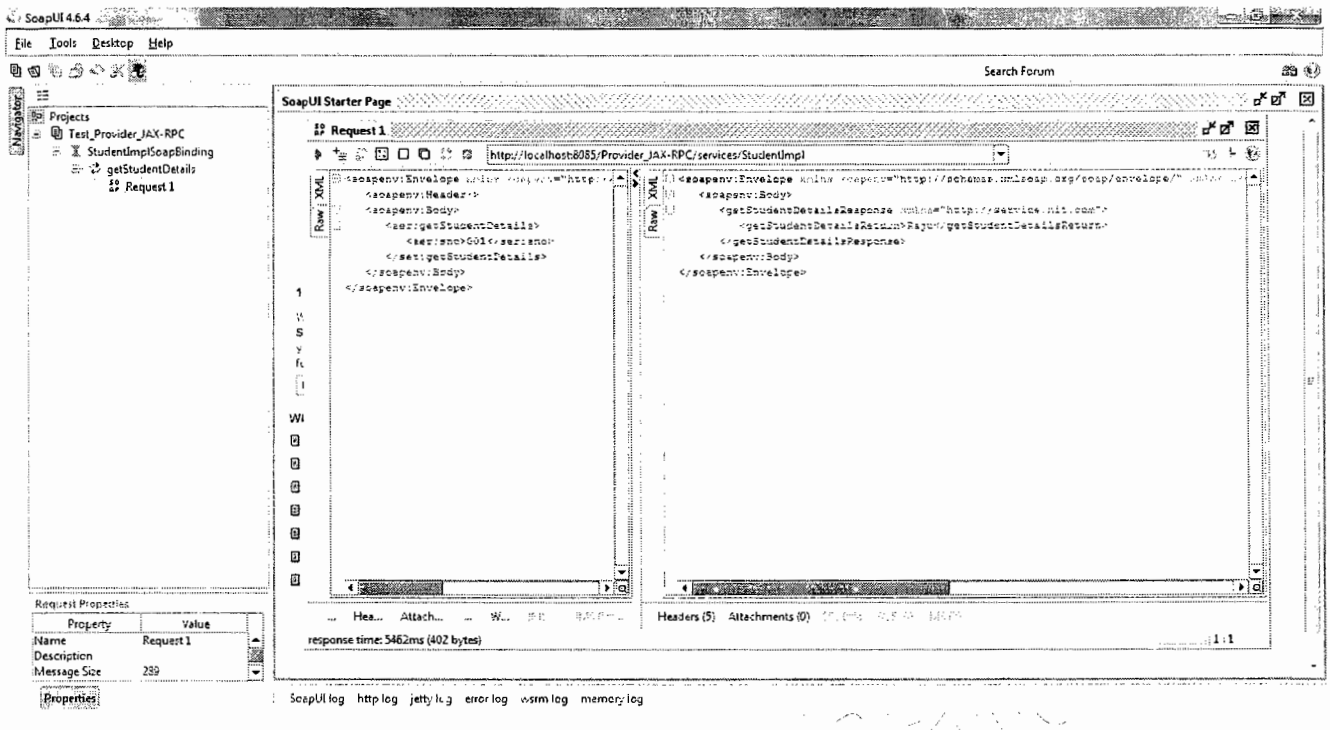
Step 3: File -->New SOAP Project



Step 4: Enter the project name.
Step 5: Browse the URL of WSDL and click on Browse button.

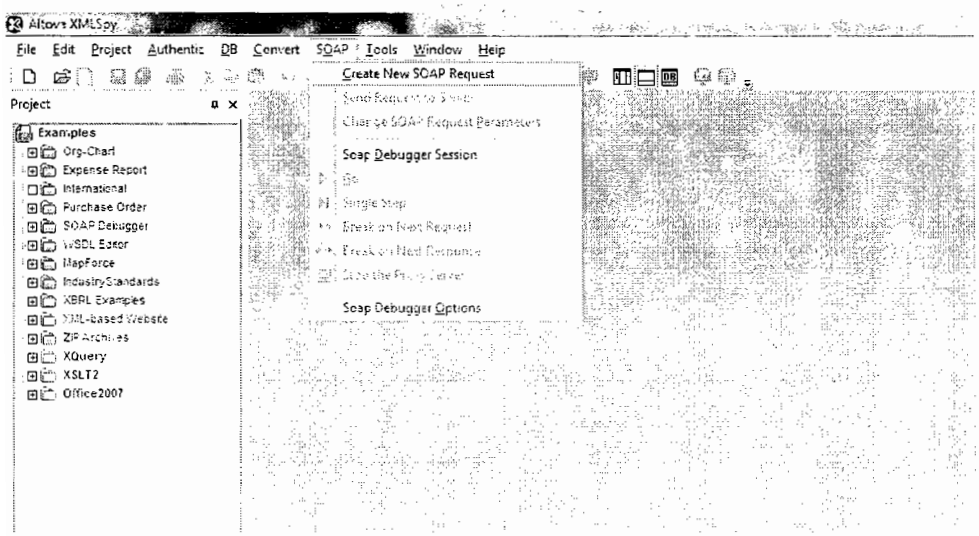


Step 6: Click on Request 1
Step 7: Observe the SOAP Request object
Step 8: Enter the Required input details
Step 9: Click on Run(Green Color) Button.
Step 10: Observe the SOAP Response with Result.

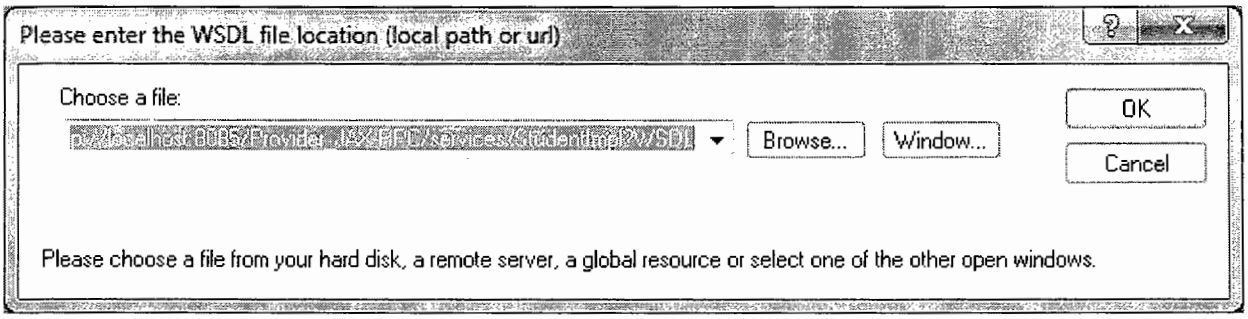


2.Altova XML Spy:

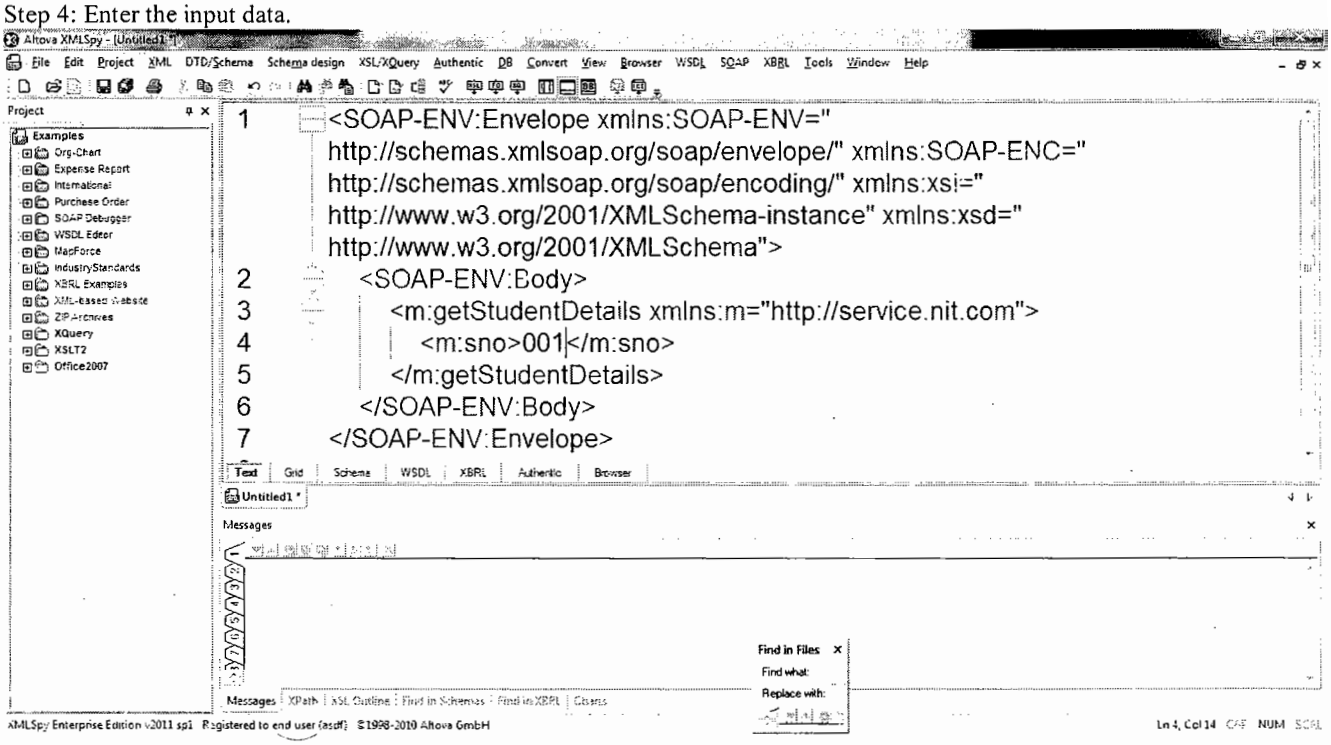
Step 1: Open Alota XML Spy.
Step 2: Create New SOAP Request.



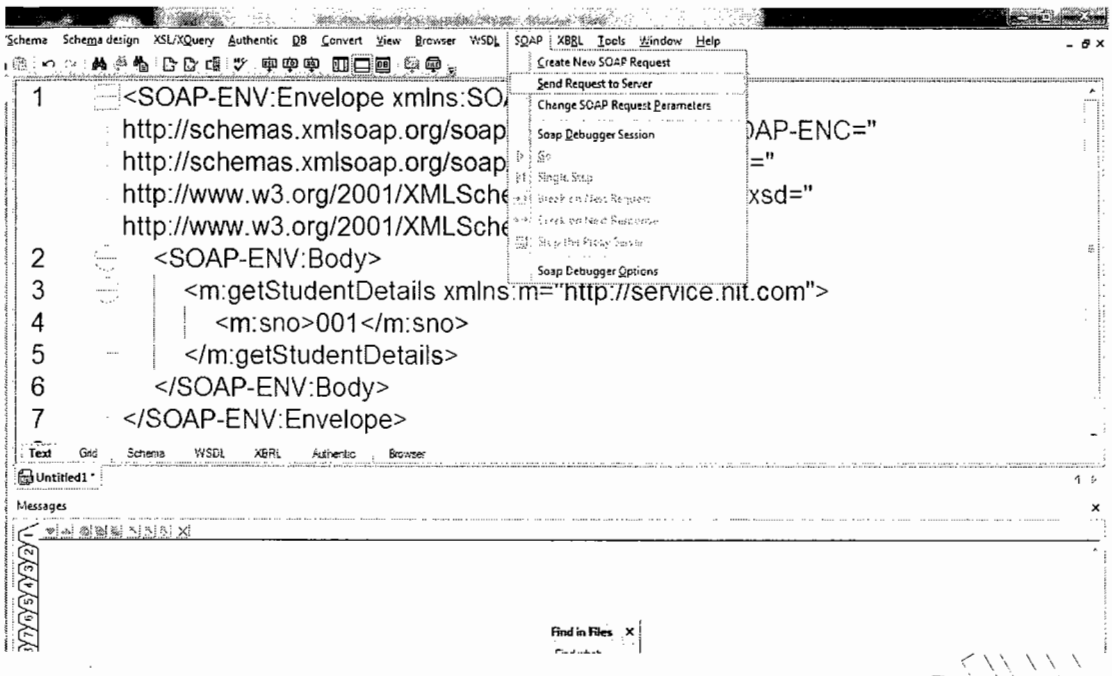
Step 3: Enter the URL Of WSDL file.



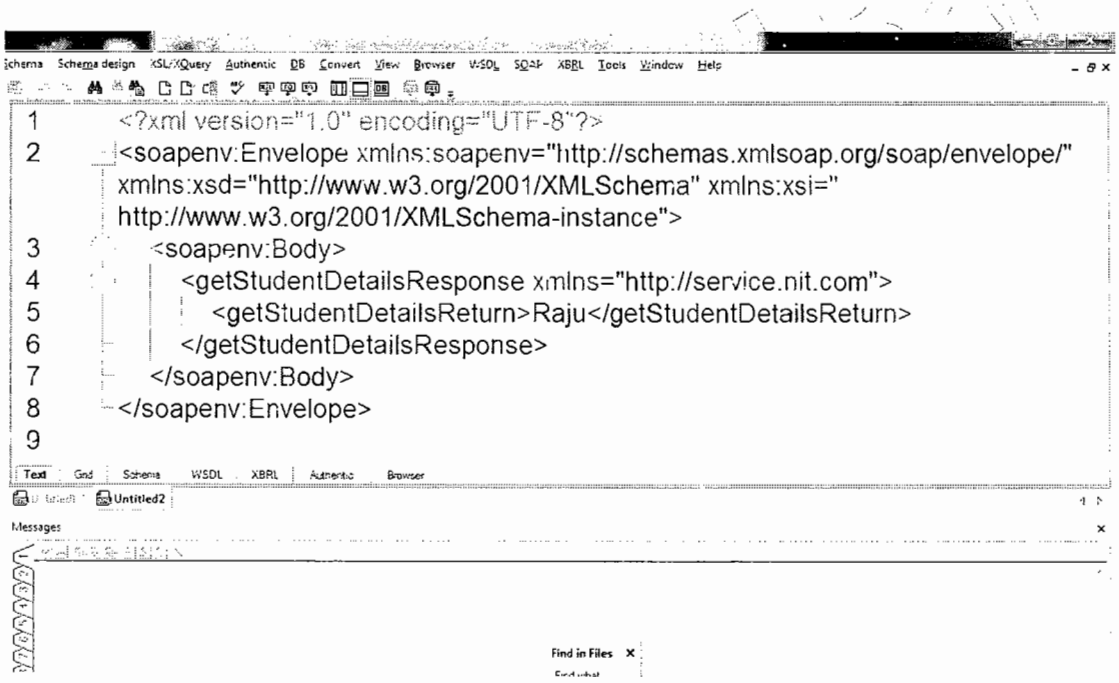
Step 4: Enter the input data.



Step 5:Click on SOAP-->Send Request to Server from tool bar.

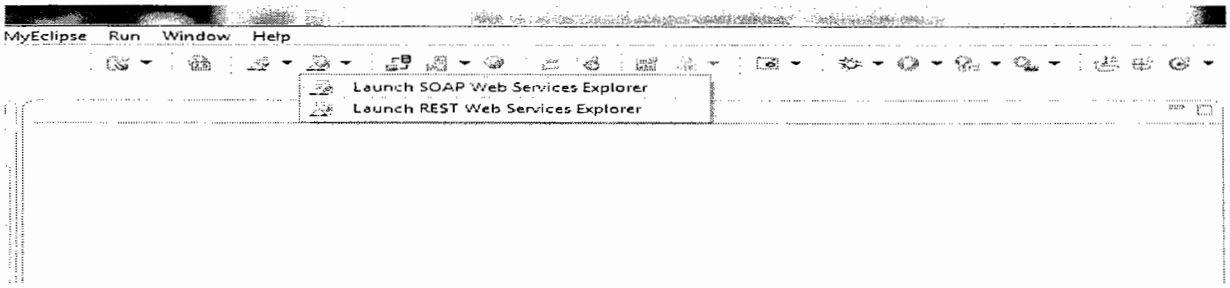


Step 6:Observe the SOAP Response:

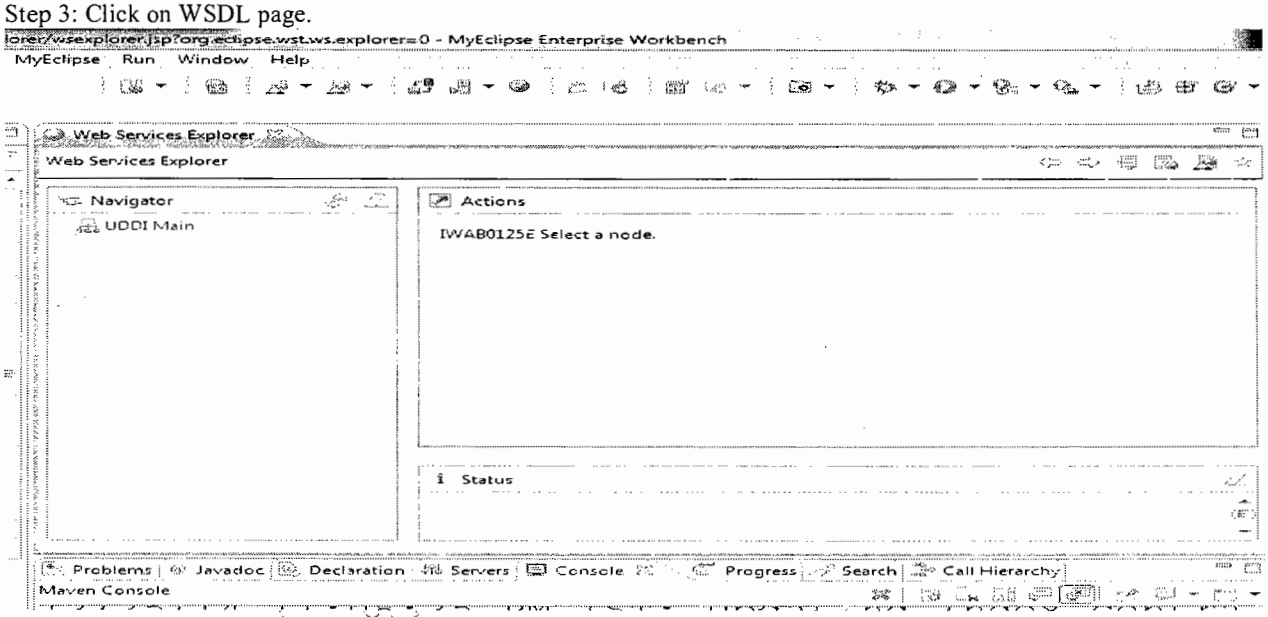


3.Launch SOAP Web Services Explorer (My Eclipse IDE)

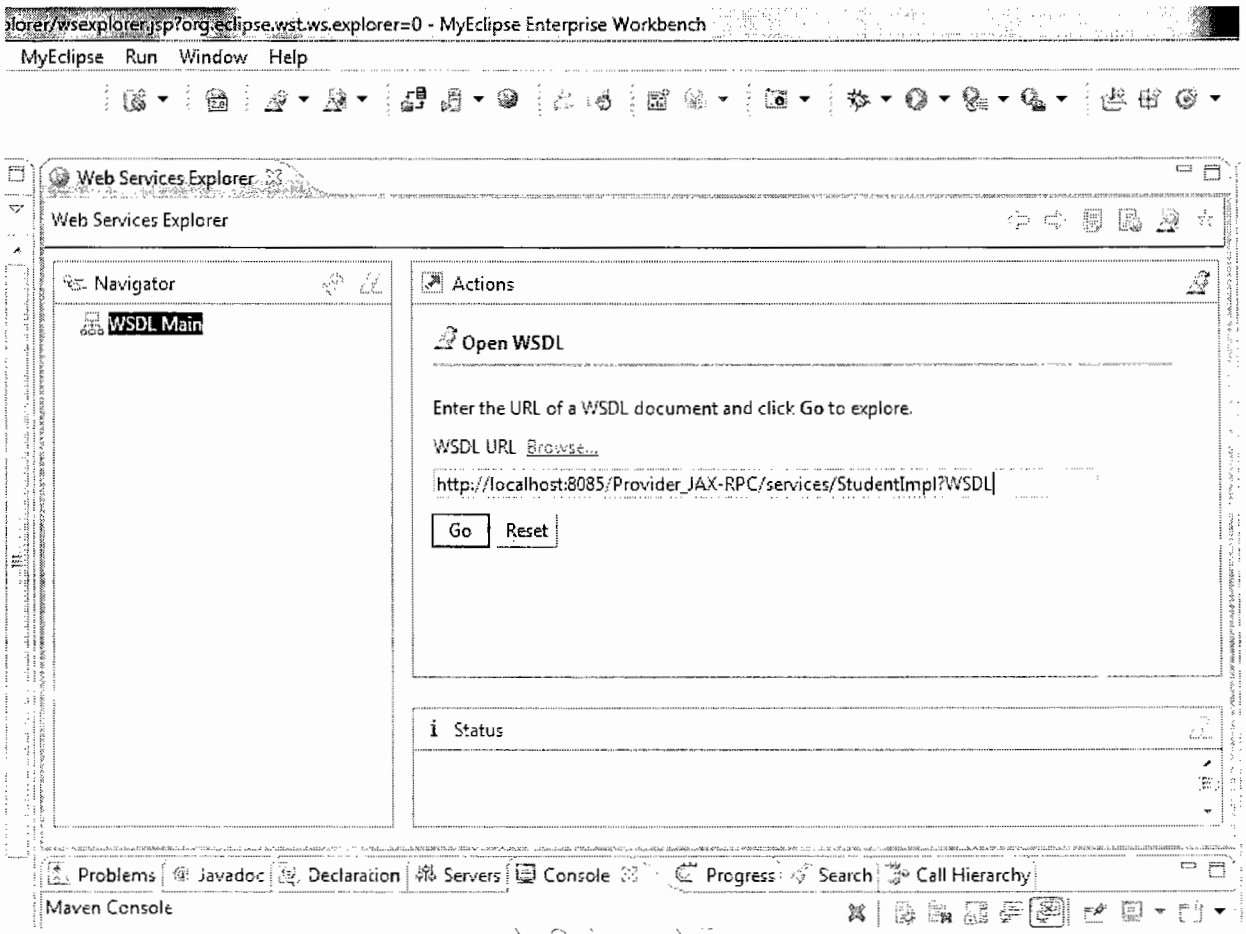
Step 1: Open the My Eclipse IDE
Step 2: Click on Launch SOAP Web Services Explorer from tool bar.



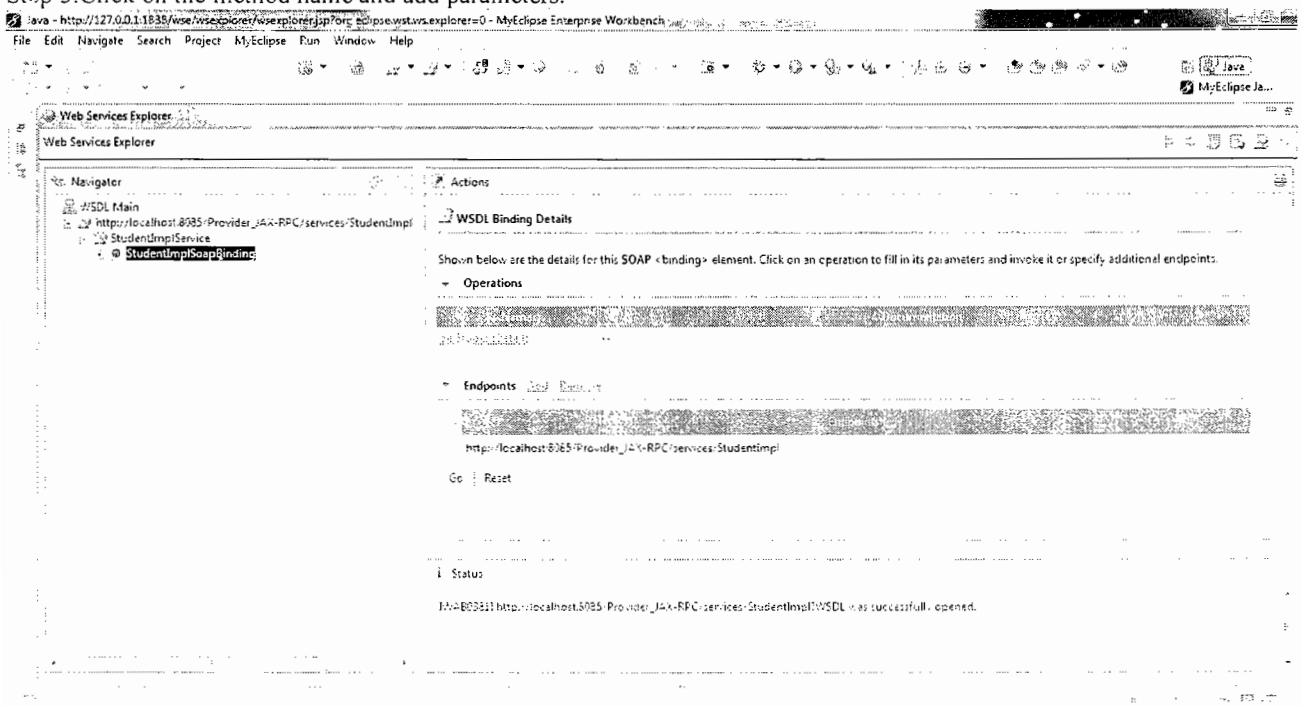
Step 3: Click on WSDL page.



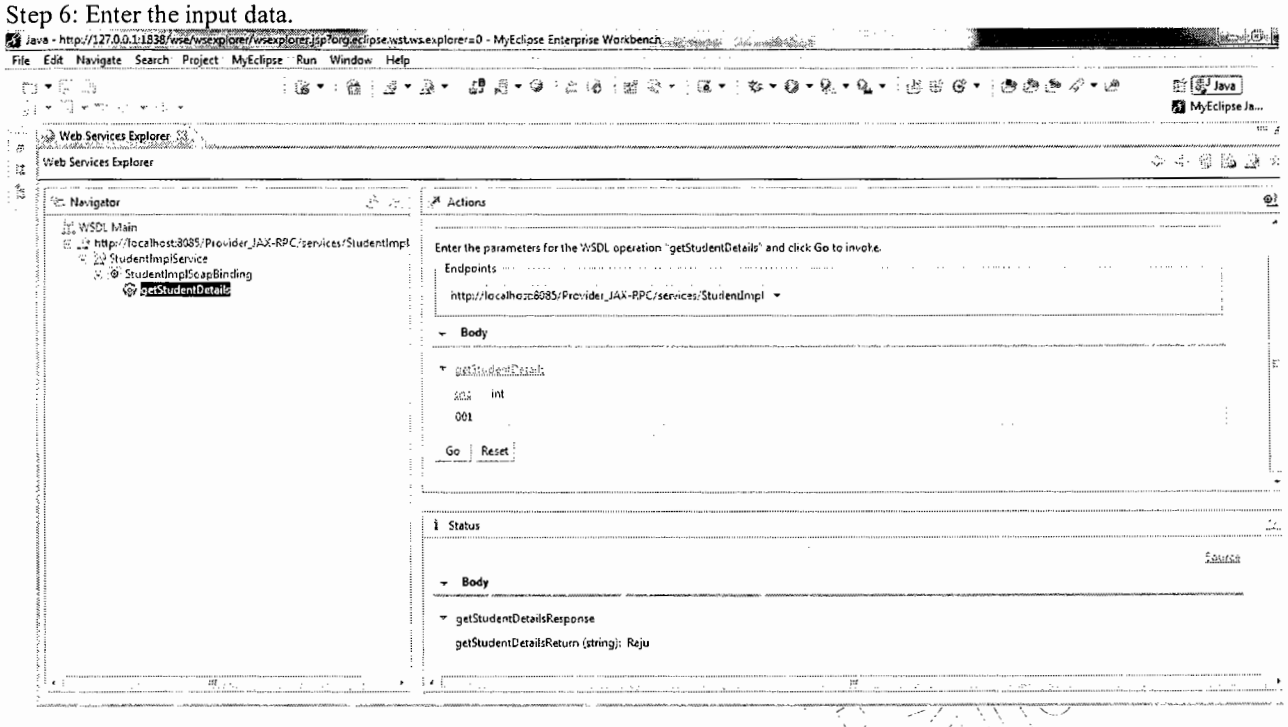
Step 4: Enter the URL of WSDL.



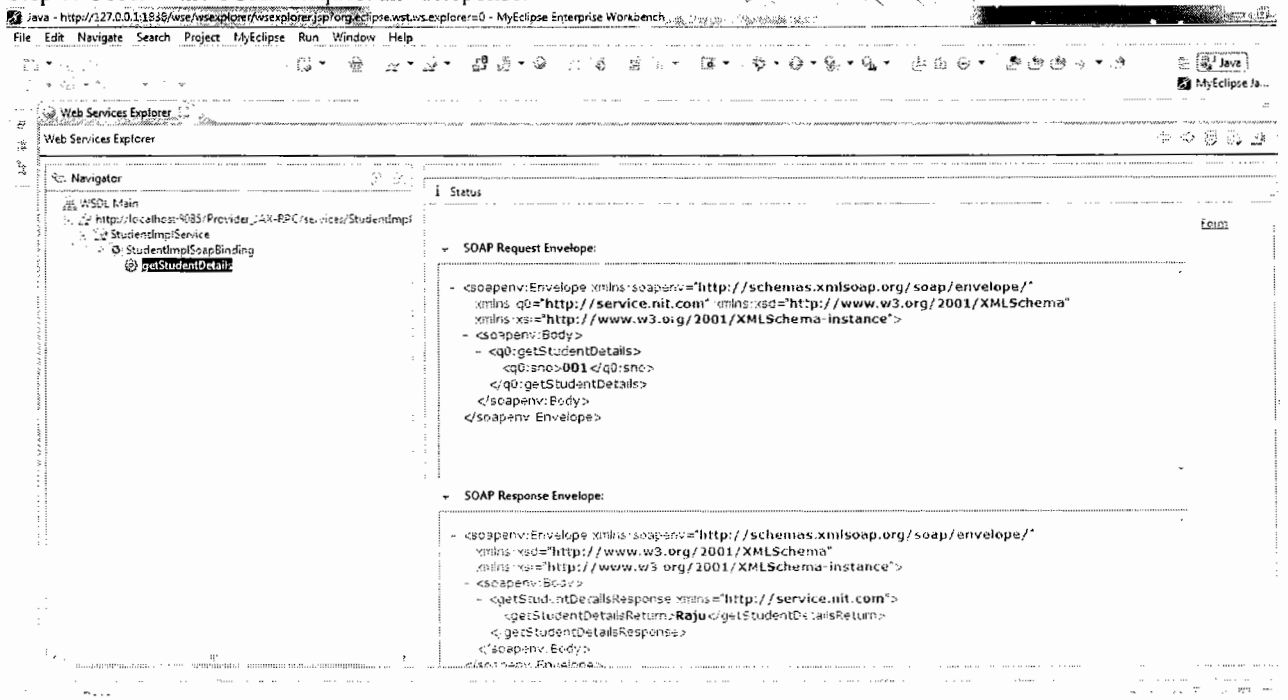
Step 5.Click on the method name and add parameters.



Step 6: Enter the input data.



Step 7: Observe the SOAP Request and Response.

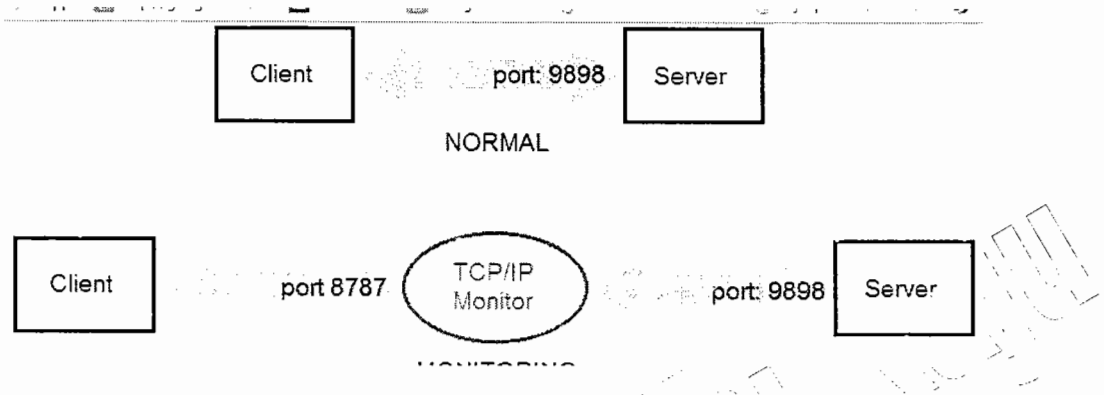


4.TCP/IP Monitor:

The Eclipse IDE has a built-in TCP/IP Monitor tool that allows programmers to intercept communications between client and server through TCP/IP or HTTP protocol in order to monitor, watch and analyze the requests and responses. This article provides the steps which help you to use the TCP/IP monitor for the purpose of monitoring HTTP communications between the client and the server on a localhost machine.

Basically, the TCP/IP monitor acts as a proxy between the client and the server so that it can capture the requests and responses. And the most important thing to remember when using this monitor is that it requires changing the server port at the client side to the port at which the monitor is listening.

The following diagram explains the concepts:



For example, suppose that we need to monitor the HTTP communications of a web application running on localhost (typically under a Tomcat instance inside Eclipse), which would be normally accessed via this URL:

http://localhost:8085/Provider_JAX-RPC/services/StudentImpl

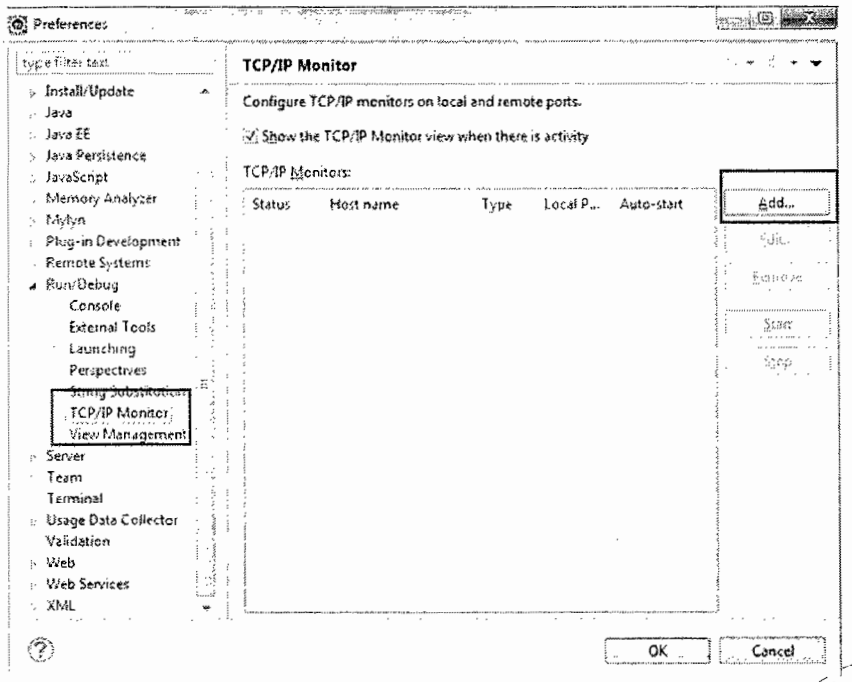
And the TCP/IP Monitor is listening on the port number 9898, therefore we have to replace the port 8085 by 9898 as follows:

http://localhost:9898/Provider_JAX-RPC/services/StudentImpl

If the monitor is running on the port 9898, it will receive the requests and forward them to the destination server at the port configured (8085).

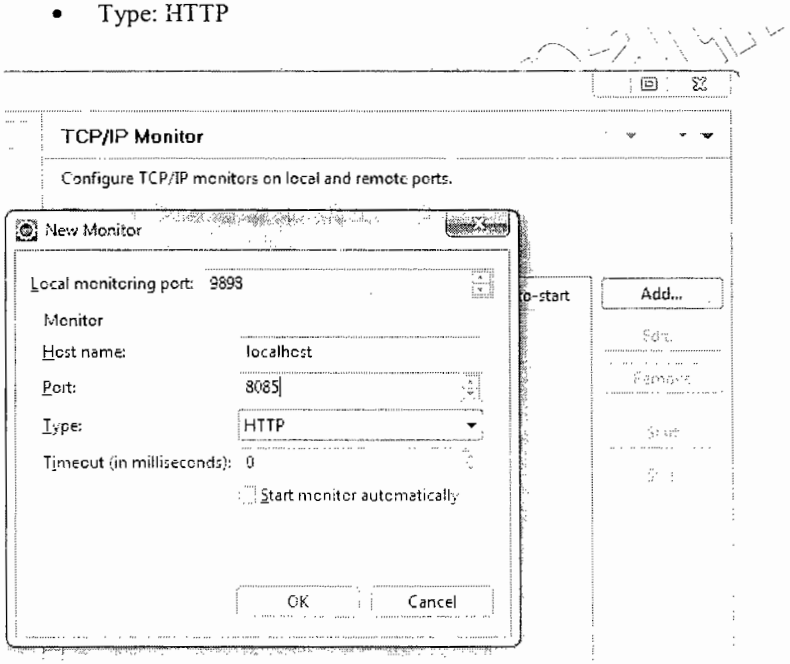
Setting up a TCP/IP Monitor in Eclipse:

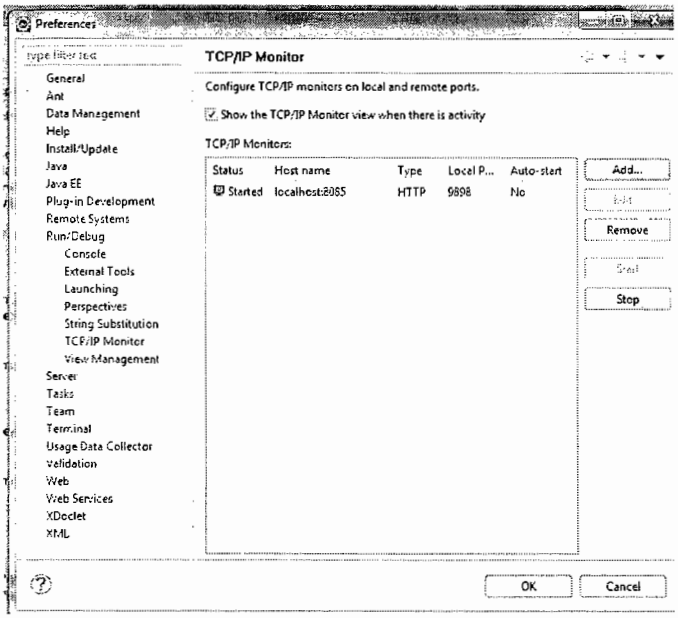
Open the TCP/IP Monitor settings by go to the menu Window > Preferences > TCP/IP Monitor:



Click Add to create a new monitor. In the New Monitor dialog, enter the following information:

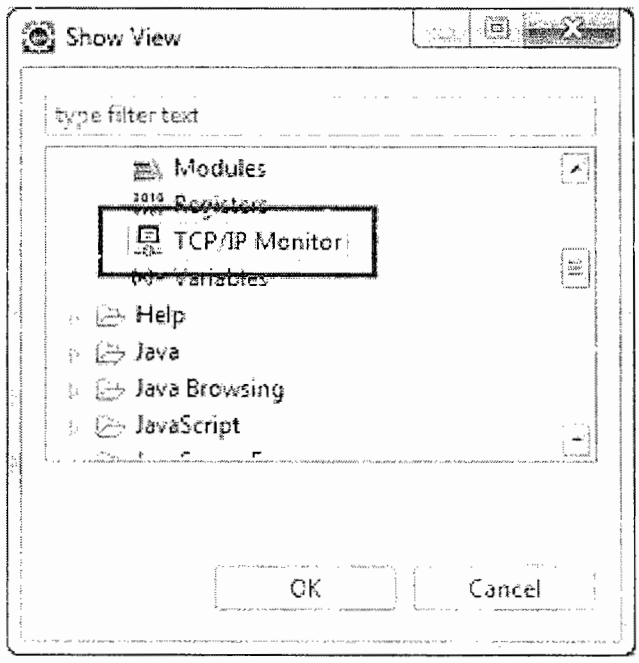
- Local monitoring port: 9898
- Host name: localhost
- Port: 8085
- Type: HTTP



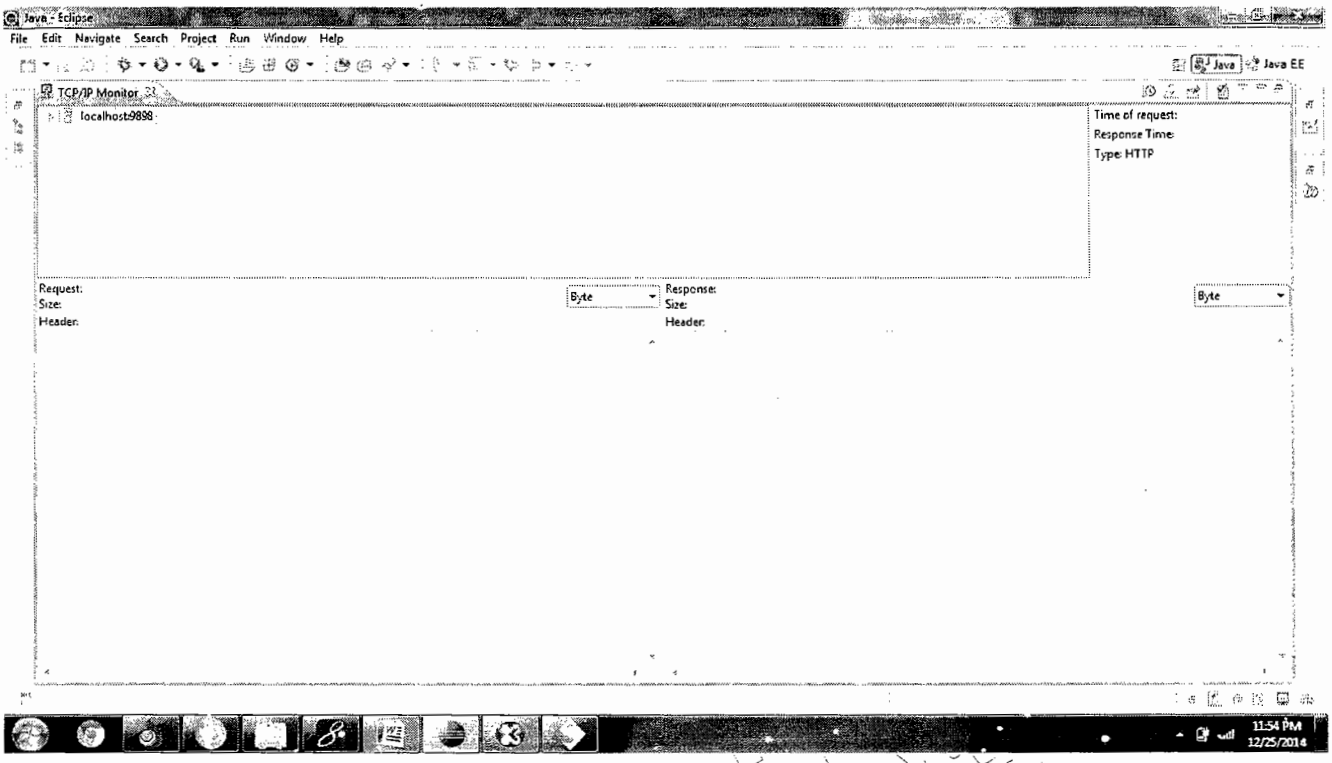


When the Status changed to Started, click OK to close the Preferences dialog. Next,

click Window > Show View > Other... > TCP/IP Monitor > OK to open up the TCP/IP Monitor view:

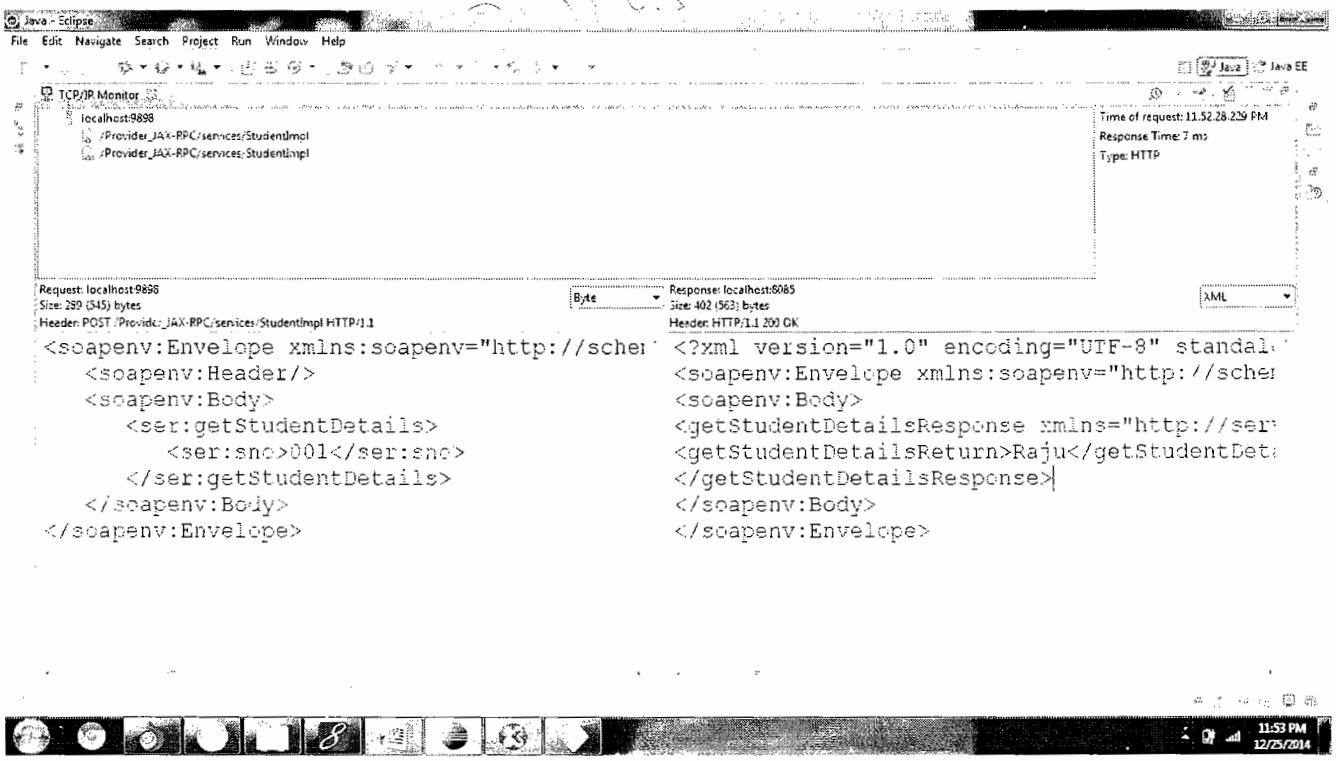


The TCP/IP Monitor view is added to the workspace but there is no information



Now, make sure that the StudentImpl Webservice application is running; open an Internal Web Browser (or an external browser) and type the following URL:

http://localhost:9898/Provider_JAX-RPC/services/StudentImp



JAX-RPC RI(Sun Implementation).

CONTRACT LAST(BOTTOM-UP):

PROVIDER:

- Code the SEI interface and implementation
- Compile the SEI and implementation class.
- Write a config.xml file.
- Use wscompile to generate the files required to deploy the service.
  wscompile -gen:server -cp build/classes -d src -keep -model model-rpc-enc.xml.gz -verbose WebContent\WEB-INF\config.xml

- Drag and drop model file and wsdl file into WEB-INF.
- write jaxrpc-ri.xml file
- Export project as war file.
- Download JWSDP 2.0 from net (which contains jaxrpc,jaxws,saaj.....)
- Use wsdeploytool to package the files into a WAR file.
  wsdeploy –o Provider_RPC_SI.war Provider_Web .war

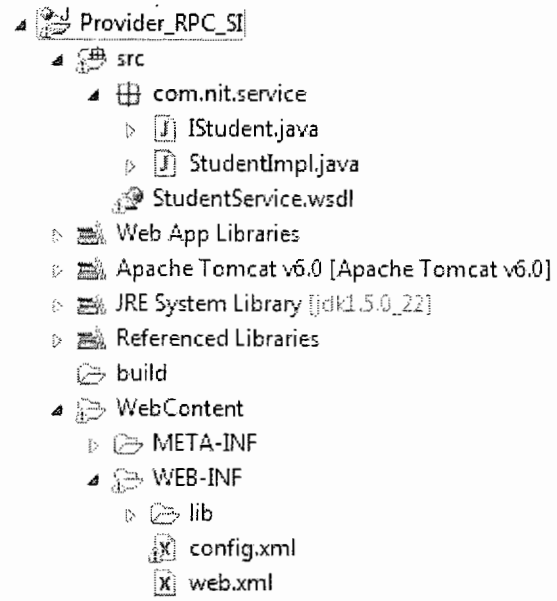
- Deploy the WAR file. The tie classes (which are used to communicate with clients) are generated by the Application Server during deployment.
- Test application

CONSUMER:

- create a java project.
- write config-client.xml
- run the command as wscompile
- wscompile -gen:client -d src -keep -verbose config-client.xml
- create a test client and run.

PROVIDER:

- Create a Dynamic Web Project.



- Code the SEI and implementation class and interface configuration file.

```
package com.nit.service;
import java.rmi.Remote;
import java.rmi.RemoteException;
public interface IStudent extends Remote{
        public String getStudentDetails(Integer sno)throws RemoteException;
}
```

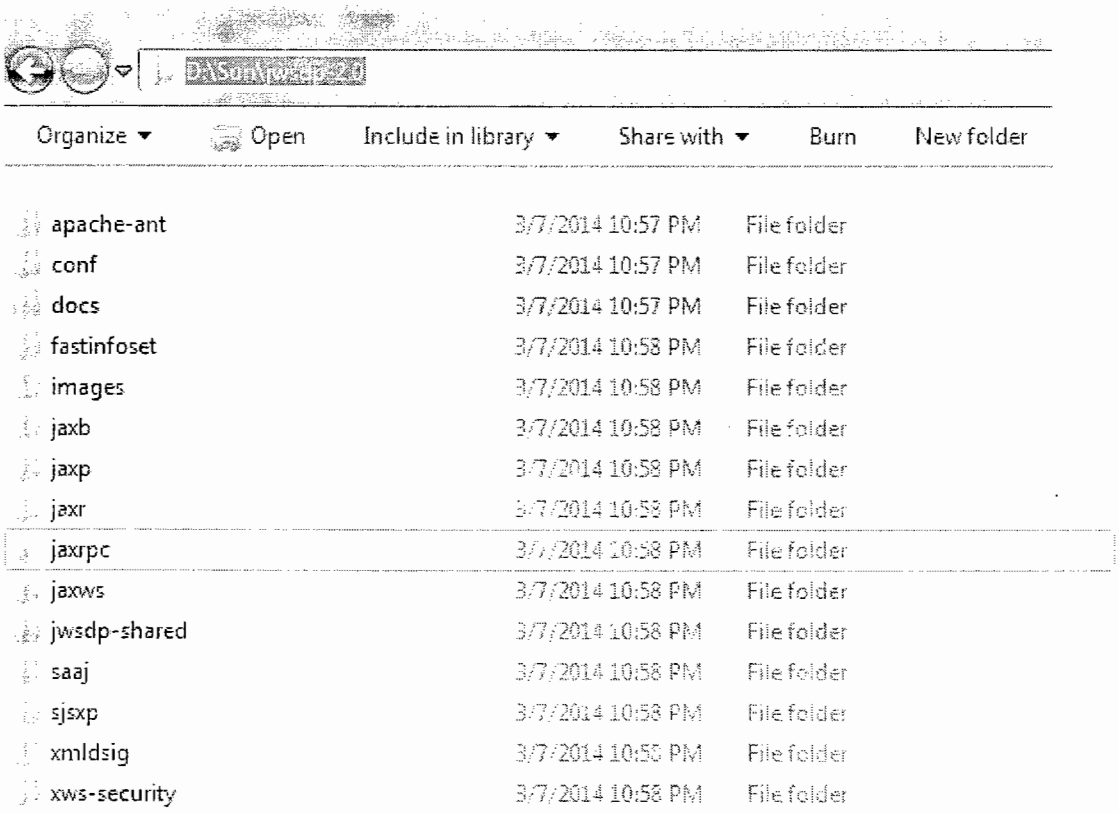
```
package com.nit.service;
import java.rmi.RemoteException;
public class StudentImpl implements IStudent {
        public String getStudentDetails(Integer sno) throws RemoteException {
                String name = null;
                if (sno == 001) {
                        name = "Raju";
                } else {
                        name = "Rani";
                }
                return name;
        }
}
```

- Compile the SEI and implementation class.
- Write a config.xml file.

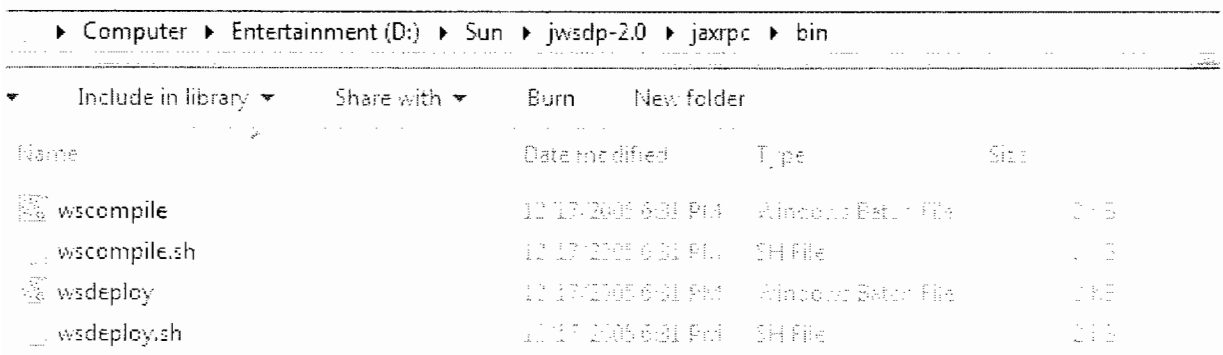
```
<?xml version="1.0" encoding="UTF-8"?>
<configuration xmlns="http://java.sun.com/xml/ns/jax-rpc/ri/config">
        <service name="StudentService"
          targetNamespace="http://rajuhelps2u.com/wsdl"
```

```
            typeNamespace="http://rajuhelps2u.com/types"
        packageName="com.nit.service.binding">
            <interface name="com.nit.service.IStudent"
                    servantName="com.nit.service.StudentImpl" />
    </service>
</configuration>
```

- Download JWSDP 2.0 from net (which contains jaxrpc,jaxws,saaj.....)



- Set the path for wscompile and wsdeploy tools in jwsdp2.0/jaxrpc/bin folder



wscompile

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

Syntax

wscompile [options] <configuration-file>

By convention, the configuration file is named config.xml, but this is not a requirement. The following table lists the wscompile options. Note that exactly one of the -import, -define, or -genoptions must be specified.

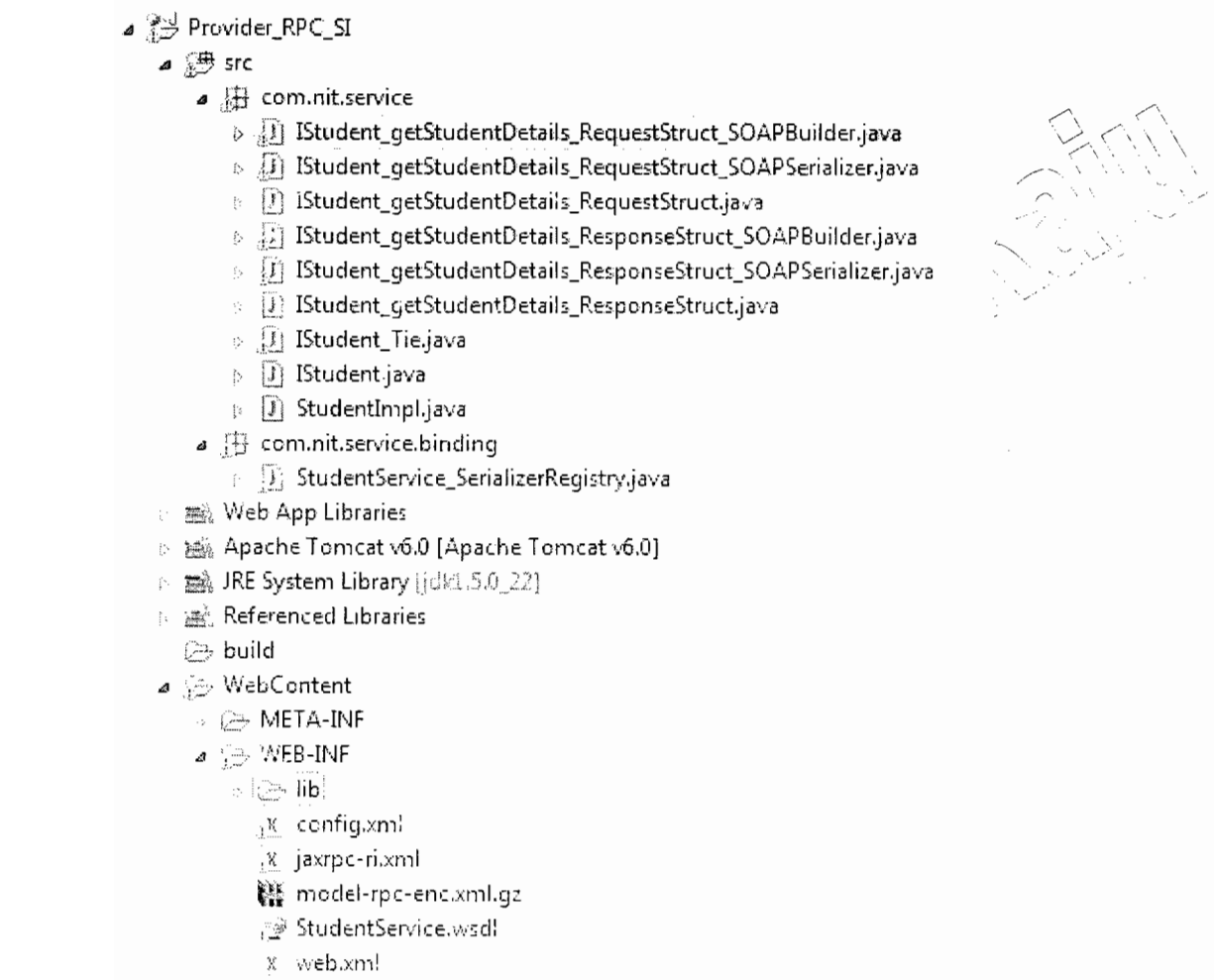
Table 1-1 wscompile Options

Option	Description
-classpath <path>	specify where to find input class files
-cp <path>	same as -classpath <path>
-d <directory>	specify where to place generated output files
-define	read the service endpoint interface, define a service
-f:<features>	enable the given features (See the below below table for a list of features. When specifying multiple features, separate them with commas.)
-features:<features>	same as -f:<features>
-g	generate debugging info
-gen	same as -gen:client
-gen:both	generate both client and server artifacts
-gen:client	generate client artifacts (stubs, etc.)
-gen:server	generate server artifacts (ties, etc.) and the WSDL file (If you are using wsdeploy you do not specify this option.)
-httpproxy:<host>:<port>	specify a HTTP proxy server (port defauits to 8080 on JWSDP)
-import	read a WSDL file, generate the service endpoint interface and a template of the class that implements the interface
-keep	keep generated files
-mapping <file>	generate a J2EE mapping.xml file (This option is not available in JWSDP.)
-model <file>	write the internal model to the given file
-nd <directory>	specify where to place non-class generated files
-O	optimize generated code
-s <directory>	specify where to place generated source files
-source <version>	Generate code for the specified JAX-RPC SI version. Supported versions are: 1.0.1, 1.0.3, and 1.1 (default).
-verbose	output messages about what the compiler is doing
-version	print version information

- Use wscompile to generate the files required to deploy the service.
  wscompile -gen:server -cp build/classes -d src -keep -model model-rpc-enc.xml.gz -verbose WebContent\WEB-INF\config.xml

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

E:\EclipseWorkspace\Naresh\Provider_RPC_SI>wscompile -gen:server -cp build/classes -d src -keep -mod
el model-rpc-enc.xml.gz -verbose WebContent\WEB-INF\config.xml
[creating model: StudentService]
[creating service: StudentService]
[creating port: com.nit.service.IStudent]
[creating operation: getStudentDetails]
[CustomClassGenerator: generating JavaClass for: getStudentDetails]
[CustomClassGenerator: generating JavaClass for: getStudentDetailsResponse]
[SOAPObjectSerializerGenerator: writing  serializer/deserializer for: {http://rajuhelps2u.com/types}
getStudentDetails]
[SOAPObjectSerializerGenerator: writing  serializer/deserializer for: {http://rajuhelps2u.com/types}
getStudentDetailsResponse]
[SOAPObjectBuilderGenerator: writing object builder for: getStudentDetails]
[SOAPObjectBuilderGenerator: writing object builder for: getStudentDetailsResponse]
[SerializerRegistryGenerator: creating serializer registry: com.nit.service.binding.StudentService_S
erializerRegistry]
E:\EclipseWorkspace\Naresh\Provider_RPC_SI>_
```

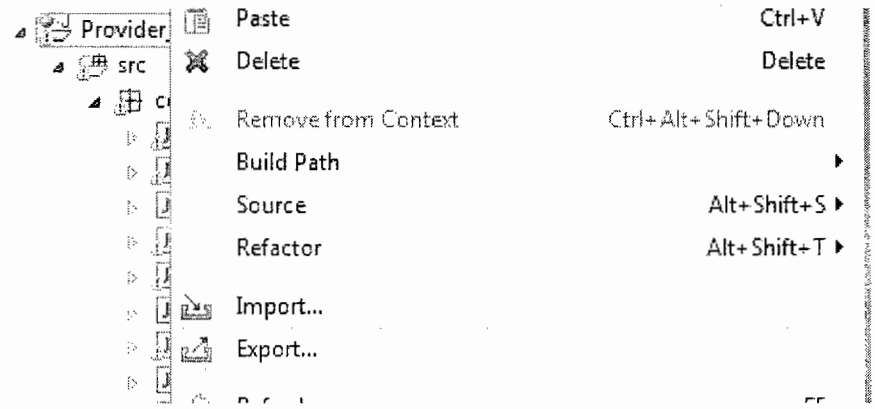
- Drag and drop 'model' file and WSDL file into WEB-INF.

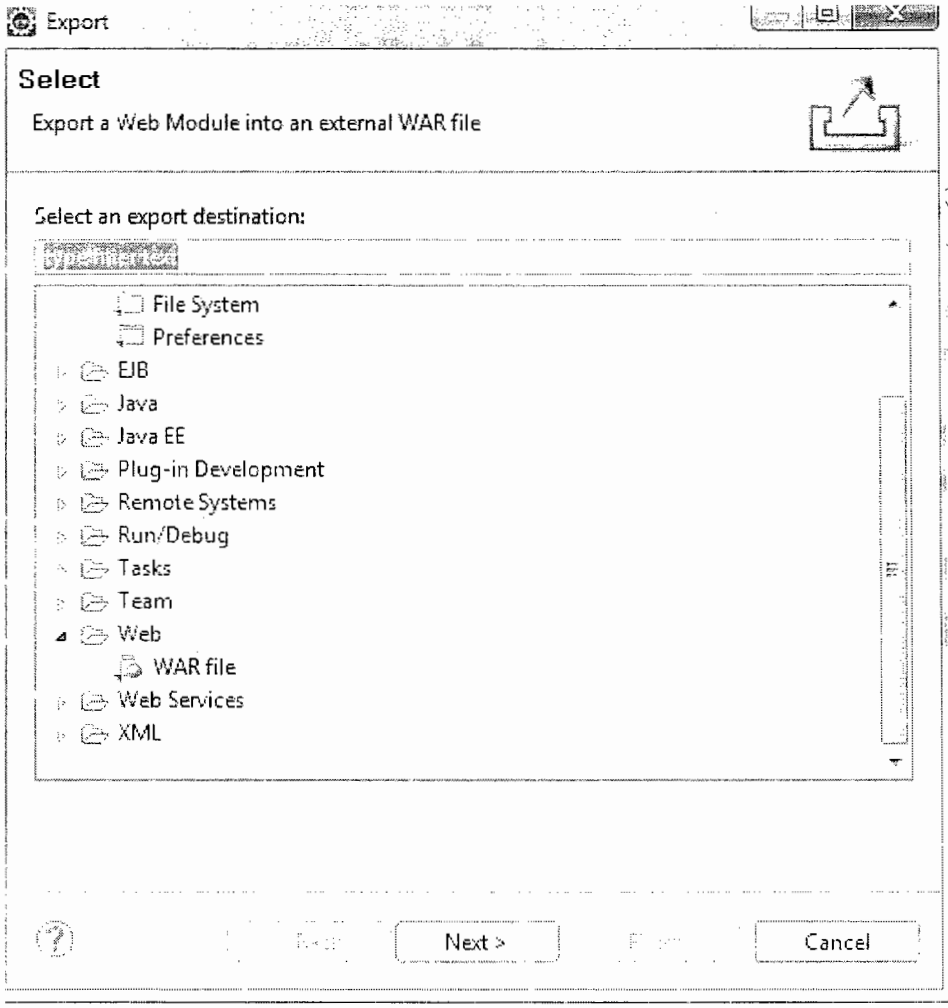


- write jaxrpc-ri.xml file

```
<?xml version="1.0" encoding="UTF-8"?>
<webServices
    xmlns="http://java.sun.com/xml/ns/jax-rpc/ri/dd"
    version="1.0"
    targetNamespaceBase="http://rajuhelps2u.com/wsdl"
    typeNamespaceBase="http://rajuhelps2u.com/types"
    urlPatternBase="/student">
    <endpoint
      name="Student"
      displayName="Student Service"
      description="A simple Student service"
      wsdl="/WEB-INF/StudentService.wsdl"
      interface="com.nit.service.IStudent"
      implementation="com.nit.service.StudentImpl"
      model="/WEB-INF/model-rpc-enc.xml.gz"/>
    <endpointMapping
      endpointName="Student"
      urlPattern="/student"/>
</webServices>
```

- Export project as war file with the name of "Provider_Web.war"


Provider
src
Paste Ctrl+V
Delete Delete
Remove from Context Ctrl+Alt+Shift+Down
Build Path
Source Alt+Shift+S
Refactor Alt+Shift+T
Import...
Export...


Export
Select
Export a Web Module into an external WAR file
Select an export destination:
File System
Preferences
EJB
Java
Java EE
Plug-in Development
Remote Systems
Run/Debug
Tasks
Team
Web
WAR file
Web Services
XML
Next >
Cancel

wsdeploy:

The `wsdeploy` tool reads a WAR file and the `jaxrpc-ri.xml` file and then generates another WAR file that is ready for deployment. Behind the scenes, `wsdeploy` runs `wscompile` with the `-gen:server` option. The `wscompile` command generates classes and a WSDL file that `wsdeploy` includes in the generated WAR file.

On JWSDP, the `wsdeploy` tool must be run. However, on J2EE you don't have to run `wsdeploy` because the functions it performs are done automatically when you deploy a WAR with `deploytool`or `asadmin`.

Syntax:

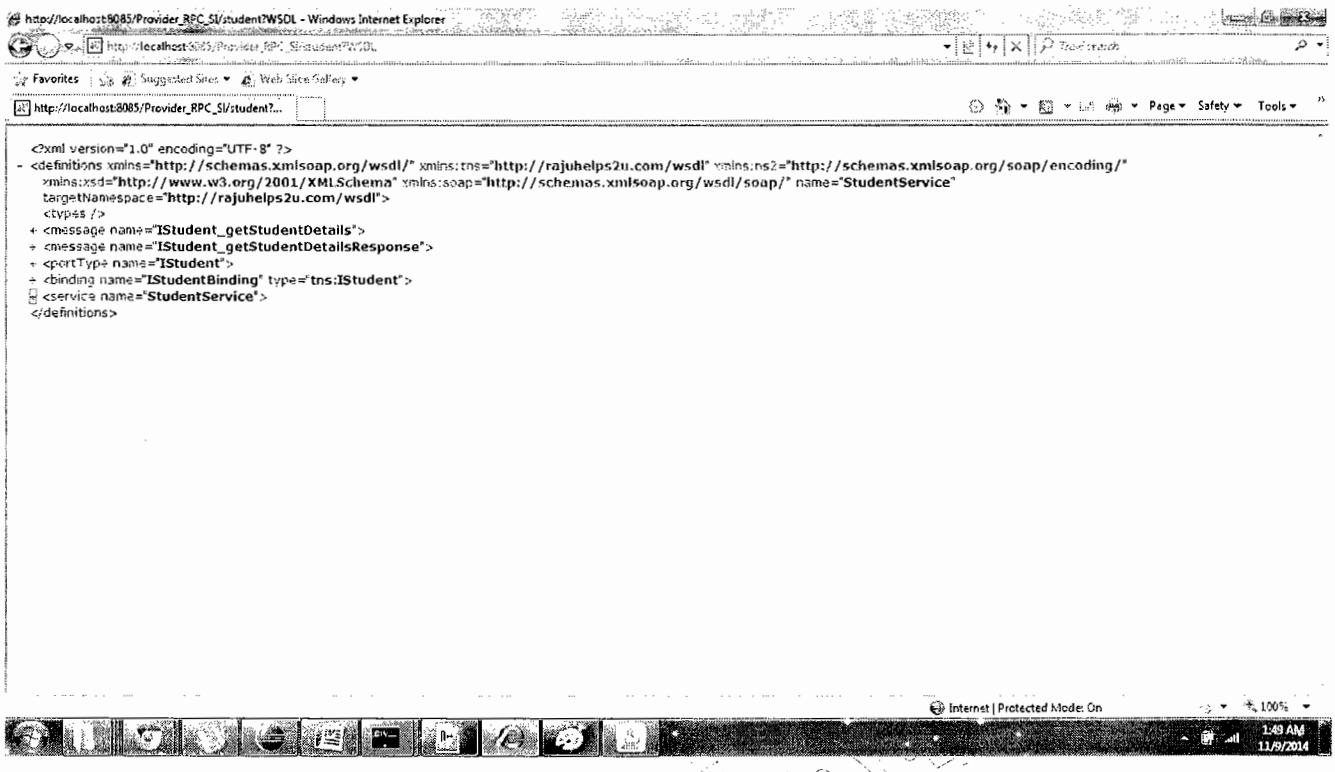
wsdeploy -o <output-war-file> <input-war-file> <options>

The following table lists the tool's options. Note that the `-o` option is required.

Option	Description
-classpath <path>	specify an optional classpath
-keep	keep temporary files
-o <output WAR file>	specify where to place the generated WAR file
-source <version>	Generate code for the specified JAX-RPC SI version. Supported versions are: 1.0.1, 1.0.3, and 1.1 (default).
-tmpdir <directory>	specify the temporary directory to use
-verbose	output messages about what the compiler is doing
-version	print version information

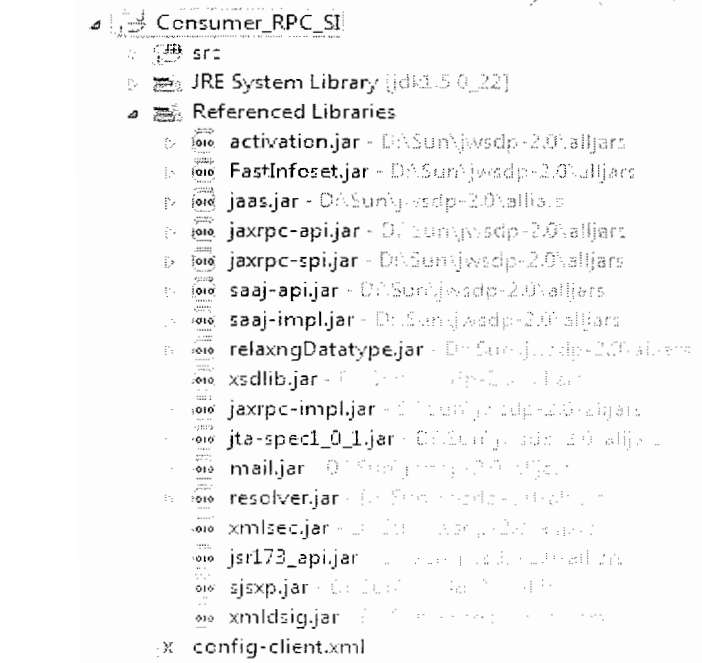
- Use wsdeploytool to package the files into a WAR file.
  wsdeploy –o Provider_RPC_SI.war Provider_Web.war
- Deploy the WAR file into server and start the server.
- Test application with the below URL

  http://localhost:8085/Provider_RPC_SI/student?WSDL



CONSUMER:

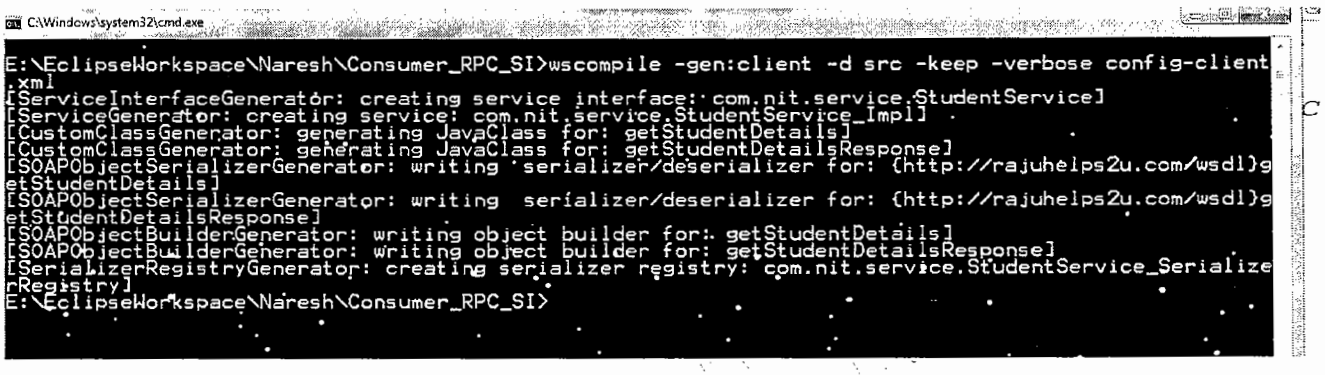
- Create a java project with the below name.



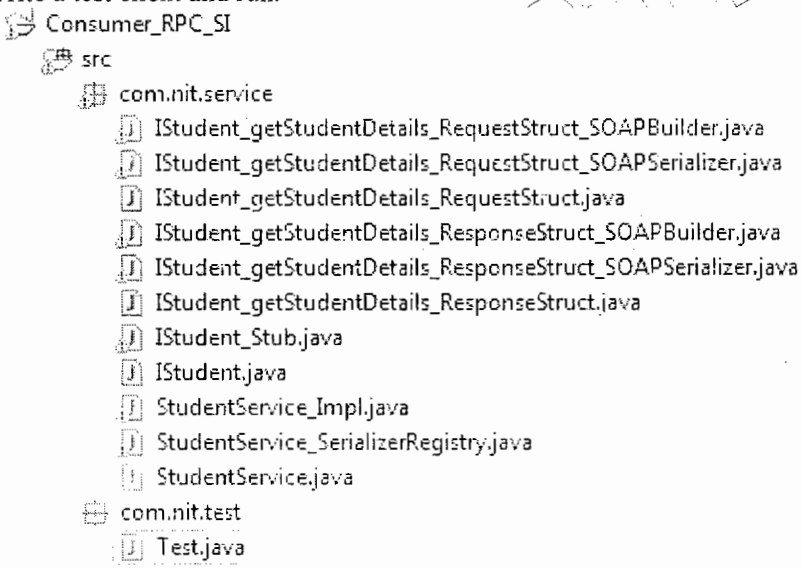
- write config-client.xml

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

- run the command as wscompile
  wscompile -gen:client -d src -keep -verbose config-client.xml



- write a test client and run.



Test.java

```
package com.nit.test;
import java.rmi.RemoteException;
import javax.xml.rpc.ServiceException;
import com.nit.service.IStudent;
import com.nit.service.StudentService;
```

```
import com.nit.service.StudentService_Impl;
public class Test {
        public static void main(String[] args) throws ServiceException, RemoteException {
                StudentService service =new StudentService_Impl();
                IStudent student =service.getIStudentPort();
                System.out.println(student.getStudentDetails(001));
        }}
```

Contract First Approach(TOP-DOWN).

PROVIDER:

- Create a web project
- Add JAX-RPC libraries.
- Place the WSDL file in WEB-INF folder.
- Write a config.xml file.
- Use wscompile to generate the files required to deploy the service.
  wscompile -gen:server -cp WebRoot/WEB-INF/classes -d src

  -keep -model model-rpc-enc.xml.gz -verbose WebRoot\WEB-INF\config.xml
- Develop an SEI Implementation class.
- write jaxrpc-ri.xml file
- Export project as war file.
- Use wsdeploytool to package the files into a WAR file.
  wsdeploy –o target.war StudentWeb.war
- Unzip target.war, copy web.xml and jaxrpc-ri-runtime.xml into WEB-INF folder.
- Test application

Config.xml

It is used to generate binding classes and stub/skeleton information which performs the handshaking process.

Placed under WEB-INF directory.

```
<configuration   xmlns="http://java.sun.com/xml/ns/jax-rpc/ri/config">
  <wsdl     location="http://localhost:8085/Provider_RPC_SI/student?WSDL"
    packageName="com.nit.service">
  </wsdl>
</configuration>
```

CONSUMER:

Steps:

1) create a java project
2) write a config.xml by giving an WSDL location.
3) Run wscompile tool.
4) wscompile -gen:client -d src -keep -verbose resources\config.xml
5) create a test package and write a client logic in java.

WebService Clients:

1. Using Generated Stub
2. Using Dynamic Proxies
3. Using Dynamic Invocation Interface (DII)

In Generated Stub, client depends on generated interfaces of service (last block in WSDL) and portType (business definition), and their implemented stub & proxy classes. The code need to create the specific service from generated stub, and get its specific portType object from it. Then call business method

In Dynamic Invocation Interface, client does not care about the generated service stub, but creates the framework "Service" object by passing the actual service name. From the framework Service object, the client gets the generated portType object by passing the portType name. Then call the business method on the portType object

In Dynamic Invocation Interface (DII), the client does not use any generated class. It creates a Service object, and creates a Call object out of it. To the call object, client sets the name of Service, name of portType and invoke it by passing an Object array of parameters.

Generated stub (Static stub)	Dynamic proxy	Dynamic Invocation Interface (DII)
Web service not expected to change	Some changes to the Web Service expected, such as the location of the service	Considerable changes to the Web service expected, such as: • Location of the service • Request/response format • Data types
Most common scenario	Less common	Less common
1.You can generate a stub class either from WSDL (using WSDL2Java) or from a service endpoint interface. 2. A generated stub class is required to implement both javax.xml.rpc.Stub and the service endpoint interface. 3.This stub interface provides APIs to configure stubs by setting properties like endpoint address, session, user name, password, etc.	1. The client at runtime creates dynamic proxy stubs using the javax.xml.rpc.Service interface. 2. The client has a prior knowledge of the WSDL and the service it is going to invoke. 3. It uses the javax.xml.rpc.ServiceFactory classes to create the service and get the proxy.	1.This software pattern eliminates the need for clients to know in advance a service's exact name and parameters. 2.A DII client can discover this information at runtime using a service broker that can look up the service's information. 3.This flexibility in service discovery enables the run-time system to use service brokers, which can adopt varying service discovery mechanisms - registries, UDDI, etc.

1. Using Generated Stub

```
package com.nit.test;
import java.rmi.RemoteException;
import javax.xml.rpc.ServiceException;
import com.nit.service.StudentImpl;
import com.nit.service.StudentImplServiceLocator;
public class Proxy_Stub {
```

```
        public static void main(String[] args) throws RemoteException, ServiceException {
                // TODO Auto-generated method stub
                StudentImplServiceLocator serviceLocator =new StudentImplServiceLocator();
                StudentImpl impl=serviceLocator.getStudentImpl();
                String name =impl.getStudentDetails(001);
                System.out.println("RESULT:"+name);
        }
}
```

2. Using Dynamic Proxies:

```
package com.nit.test;
import java.net.URL;
import java.rmi.RemoteException;
import javax.xml.namespace.QName;
import javax.xml.rpc.Service;
import javax.xml.rpc.ServiceException;
import org.apache.axis.client.ServiceFactory;
import com.nit.service.StudentImpl;

public class DynamicProxies {
        public static void main(String[] args) throws RemoteException, ServiceException {
                try{
                ServiceFactory serviceFactory = (ServiceFactory) ServiceFactory.newInstance();
                System.out.println("Got the service factory");

                String endpoint = "http://localhost:8085/Provider_RPC/services/StudentImpl?wsdl";
                URL wsdlURL = new URL(endpoint);

                Service service = (Service) serviceFactory.createService(
                wsdlURL, new QName("http://service.nit.com", "StudentImplService"));
                System.out.println("Got the service: " + service);

                StudentImpl impl = (StudentImpl) service.getPort(
                new QName("http://service.nit.com", "StudentImpl"), StudentImpl.class);

                System.out.println( impl.getStudentDetails(001));
                } catch (Exception e) {
                System.err.println(e.toString());
                }
        }
}
```

3. Using Dynamic Invocation Interface (DII)

```
package com.nit.test;
import java.rmi.RemoteException;
import javax.xml.namespace.QName;
import javax.xml.rpc.ServiceException;
import org.apache.axis.client.Call;
import org.apache.axis.client.Service;
public class DynamicInvocationInterface {
        public static void main(String[] args) throws RemoteException,
                        ServiceException {
                try {
```

```
                String endpoint = "http://localhost:8085/Provider_RPC/services/StudentImpl";
                Service service = new Service();
                Call call = (Call) service.createCall();
                call.setTargetEndpointAddress(new java.net.URL(endpoint));
                call.setOperationName(new QName("getStudentDetails"));
                String result = (String) call.invoke(new Object[] { "001" });
                System.out.println("Result:" + result);
            } catch (Exception e) {
                System.err.println(e.toString());
            }
        }
}
```

AXIS

Apache Axis is an Open Source SOAP server and client. SOAP is a mechanism for inter-application communication between systems written in arbitrary languages, across the Internet. SOAP usually exchanges messages over HTTP: the client POSTs a SOAP request, and receives either an HTTP success code and a SOAP response or an HTTP error code. Open Source means that you get the source,

SOAP messages are XML messages. These messages exchange structured information between SOAP systems. Messages consist of one or more SOAP elements inside an envelope, Headers and the SOAP Body. Axis handles the magic of converting Java objects to SOAP data when it sends it over the wire or receives results. SOAP Faults are sent by the server when something goes wrong; Axis converts these to Java exceptions.

Axis implements the JAX-RPC API, one of the standard ways to program Java services.Axis also provides extension features that in many ways extends the JAX-RPC API. You can use these to write better programs, but these will only work with the Axis implementation

Axis is compiled in the JAR file axis.jar; it implements the JAX-RPC API declared in the JAR files jaxrpc.jar and saaj.jar. It needs various helper libraries, for logging, WSDL processing and introspection. All these files can be packaged into a web application, axis.war, that can be dropped into a servlet container. Axis ships with some sample SOAP services. You can add your own by adding new compiled classes to the Axis webapp and registering them.

Before you can do that, you have to install it and get it working.

PROVIDER

1. Download Axis from axis.apache.org. i.e. **axis-bin-1_3.zip**.

2. Copy **axis** directory of **d:\axis-1_3\webapps** into **webapps** directory of **Tomcat5**.

3. Start Tomcat and test whether the Axis web application is running using **http://localhost:8080/axis** .

4.Test whether Axis has access to all required libraries by using **http://localhost:8080/axis/happyaxis.jsp**. If any core libraries are missing, make sure you copy them to either **WEB-INF/lib** directory of Axis or **common\lib** directory of Tomcat.

5. Creating a StudentService.java

```
public class StudentService
 {
  public String getStudentDetails(Integer sno){
     String name =null;
     if(sno ==001){
      name ="Raju";
     }
     return name;
  }
 }
```

E:\Java\WebService Examples>set classpath=D:\Installed Softwares\axis-bin-1_3\axis-1_3\lib\axis.jar;D:\Installed Softwares\axis-bin-1_3\axis-1_3\lib\log4j-1.2.8.jar;D:\Installed Softwares\axis-bin-1_3\axis-1_3\lib\jaxrpc.jar;D:\Installed Softwares\axis-bin-1_3\axis-1_3\lib\commons-logging-1.0.4.jar;D:\Installed Softwares\axis-bin-1_3\axis-1_3\lib\commons-discovery-0.2.jar;.;

6.Compile StudentService.java and copy StudentService.class into WEB-INF/classes directory of axis application in Tomcat.
7.Create StudentService.wsdd , which contains details of the service as follows:

```
<deployment name="test" xmlns="http://xml.apache.org/axis/wsdd/"
xmlns:java="http://xml.apache.org/axis/wsdd/providers/java">
 <service name="student" provider="java:RPC">
  <parameter name="className" value="StudentService"/>
   <parameter name="allowedMethods" value="*"/>
 </service>
</deployment>
```

8.Run AdminClient to deploy service in StudentService.wsdd as follows:

```
java org.apache.axis.client.AdminClient -lhttp://localhost:8080/axis/services/AdminService StudentService.wsdd
```

9.Test web service by using
http://localhost:8080/axis/services/student

CONSUMER:

1. Write a client code compile and run.

Client.java

```
import org.apache.axis.AxisFault;
import org.apache.axis.client.Call;
import org.apache.axis.client.Service;
import org.apache.axis.encoding.XMLType;
import org.apache.axis.utils.Options;
import javax.xml.namespace.QName;
        import javax.xml.rpc.ParameterMode;
        import java.net.URL;
        public class Client          {
         public static void main(String args[])  throws Exception
         {
          Service  service = new Service();
          Call    call   = (Call) service.createCall();
          call.setTargetEndpointAddress( "http://localhost:8080/axis/services/student");
          call.setOperationName("getStudentDetails");
          Object ret = call.invoke( new Object[] {001} );
          String res  = (String) ret;
          System.out.println(res);
         }
        }
```

WSDL

WSDL

WSDL stands for Web Services Description Language

- WSDL is an XML based protocol for information exchange in decentralized and distributed environments.
- WSDL is the standard format for describing a web service.
- WSDL definition describes how to access a web service and what operations it will perform.
- WSDL is a language for describing how to interface with XML-based services.
- WSDL is an integral part of UDDI, an XML-based worldwide business registry.
- WSDL is the language that UDDI uses.
- WSDL was developed jointly by Microsoft and IBM.
- WSDL is pronounced as 'wiz-dull' and spelled out as 'W-S-D-L'

WSDL breaks down Web services into three specific, identifiable elements that can be combined or reused once defined.

A WSDL document has various elements, but they are contained within these three main elements, which can be developed as separate documents and then they can be combined or reused to form complete WSDL files.

Following are the elements of WSDL document. Within these elements are further subelements, or parts:

- **Definition:** element must be the root element of all WSDL documents. It defines the name of the web service, declares multiple namespaces used throughout the remainder of the document, and contains all the service elements described here.
- **Data types:** the data types - in the form of XML schemas or possibly some other mechanism - to be used in the messages
- **Message:** an abstract definition of the data, in the form of a message presented either as an entire document or as arguments to be mapped to a method invocation.
- **Operation:** the abstract definition of the operation for a message, such as naming a method, message queue, or business process, that will accept and process the message
- **Port type :** an abstract set of operations mapped to one or more end points, defining the collection of operations for a binding; the collection of operations, because it is abstract, can be mapped to multiple transports through various bindings.
- **Binding:** the concrete protocol and data formats for the operations and messages defined for a particular port type.
- **Port:** a combination of a binding and a network address, providing the target address of the service communication.
- **Service:** a collection of related end points encompassing the service definitions in the file; the services map the binding to the port and include any extensibility definitions.

In addition to these major elements, the WSDL specification also defines the following utility elements:

- **Documentation:** element is used to provide human-readable documentation and can be included inside any other WSDL element.
- **Import:** element is used to import other WSDL documents or XML Schemas.

The WSDL Document Structure

The main structure of a WSDL document looks like this:

```
<definitions>
<types>
  definition of types.....
</types>
<message>
  definition of a message....
</message>
<portType>
    <operation>
      definition of a operation.....
    </operation>
</portType>
<binding>
  definition of a binding....
</binding>
<service>
  definition of a service....
  <port></port>
</service>
</definitions>
```

Content of StudentService.wsdl file

```
<?xml version="1.0" encoding="UTF-8" ?>
<definitions targetNamespace="http://service.nit.com" xmlns:apachesoap="http://xml.apache.org/xml-
soap" xmlns:impl="http://service.nit.com" xmlns="http://schemas.xmlsoap.org/wsdl/"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" name=" StudentImplService">
<types>
<schema elementFormDefault="qualified" targetNamespace="http://service.nit.com"
xmlns="http://www.w3.org/2001/XMLSchema">
<element name="getStudentDetails">
<complexType>
<sequence>
 <element name="sno" type="xsd:int" />
 </sequence>
 </complexType>
 </element>
<element name="getStudentDetailsResponse">
<complexType>
<sequence>
```

```
 <element name="getStudentDetailsReturn" type="xsd:string" />
 </sequence>
 </complexType>
 </element>
 </schema>
 </types>
<message name="getStudentDetailsResponse">
 <part element="impl:getStudentDetailsResponse" name="parameters" />
 </message>
<message name="getStudentDetailsRequest">
 <part element="impl:getStudentDetails" name="parameters" />
 </message>
<portType name="StudentImpl">
<operation name="getStudentDetails">
 <input message="impl:getStudentDetailsRequest" name="getStudentDetailsRequest" />
 <output message="impl:getStudentDetailsResponse" name="getStudentDetailsResponse" />
 </operation>
 </portType>
<binding name="StudentImplSoapBinding" type="impl:StudentImpl">
 <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http" />
<operation name="getStudentDetails">
 <soap:operation soapAction="" />
<input name="getStudentDetailsRequest">
 <soap:body use="literal" />
 </input>
<output name="getStudentDetailsResponse">
 <soap:body use="literal" />
 </output>
 </operation>
 </binding>
<service name="StudentImplService">
 <port binding="impl:StudentImplSoapBinding" name="StudentImpl">
 <soap:address location="http://localhost:8085/Provider_RPC/services/StudentImpl" />
 </port>
 </service>
 </definitions>
```

Analysis of the Example:

- **Definition :** StudentImplService
- **Type :** Using built-in data types and they are defined in XMLSchema.
- **Message :**
    1. getStudentDetailsRequest: input parameter
    2. getStudentDetailsResponse: getting return value
- **Port Type:** getStudentDetails **operation** that consists of a request and response service.
- **Binding:** Direction to use the SOAP HTTP transport protocol.
- **Service:** Service available at http://localhost:8085/Provider_RPC/services/StudentImpl.
- **Port:** Associates the binding with the URI http://localhost:8085/Provider_RPC/services/StudentImpl where the running service can be accessed.

1) <definition>

This element must be the root element of all WSDL documents. It defines the name of the web service.
Here is the example piece of code from last session which uses *definition* element.

```
<definitions targetNamespace="http://service.nit.com"
xmlns:apachesoap="http://xml.apache.org/xml-soap" xmlns:impl="http://service.nit.com"
xmlns="http://schemas.xmlsoap.org/wsdl/" xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" name=" StudentImplService">
```

From the above example we can conclude the followings points:

- The definitions element is a container of all the other elements.
- The definitions element specifies that this document is the StudentImplService.
- The definitions element specifies a *targetNamespace* attribute. The *targetNamespace* is a convention of XML Schema that enables the WSDL document to refer to itself. In this example we have specified a *targetNamespace* of http://service.nit.com.
- The definition element specifies a default namespace: xmlns:wsdl=http://schemas.xmlsoap.org/wsdl/. All elements without a namespace prefix, such as *message* or *portType*, are therefore assumed to be part of the default WSDL namespace.
- It also specifies numerous namespaces that will be used throughout the remainder of the document.

2)<types>

A Web service needs to define its inputs and outputs and how they are mapped into and out of services. WSDL <types> element take care of defining the data types that are used by the web service. Types are XML documents, or document parts.
Here is a piece of code taken from W3C specification. This code depicts how a types element can be used within a WSDL.

- The types element describes all the data types used between the client and server.
- WSDL is not tied exclusively to a specific typing system
- WSDL uses the W3C XML Schema specification as its default choice to define data types.
- If the service uses only XML Schema built-in simple types, such as strings and integers, then types element is not required.
- WSDL allows the types to be defined in separate elements so that the types are reusable with multiple Web services.

```
<types>
```

```
<schema elementFormDefault="qualified" targetNamespace="http://service.nit.com"
xmlns="http://www.w3.org/2001/XMLSchema">
<element name="getStudentDetails">
<complexType>
<sequence>
 <element name="sno" type="xsd:int" />
 </sequence>
 </complexType>
 </element>
<element name="getStudentDetailsResponse">
<complexType>
<sequence>
 <element name="getStudentDetailsReturn" type="xsd:string" />
 </sequence>
 </complexType>
 </element>
 </schema>
 </types>
```

3) <message>

This element describes the data being exchanged between the Web service providers and consumers.

- Each Web Service has two messages: input and output.
- The input describes the parameters for the Web Service and the output describes the return data from the Web Service.
- Each message contains zero or more **<part>** parameters, one for each parameter of the Web Service's function.
- Each **<part>** parameter associates with a concrete type defined in the **<types>** container element.

Lets take a piece of code from the Example Session:

```
<message name="getStudentDetailsResponse">
 <part element="impl:getStudentDetailsResponse" name="parameters" />
 </message>
<message name="getStudentDetailsRequest">
 <part element="impl:getStudentDetails" name="parameters" />
 </message>
```

Here, two message elements are defined. The first represents a request message getStudentDetailsRequest, and the second represents a response message getStudentDetailsResponse.

4)<portType>

This element combines multiple message elements to form a complete oneway or round-trip operation.
For example, a **<portType>** can combine one request and one response message into a single request/response operation. This is most commonly used in SOAP services. A portType can define multiple operations.

Lets take a piece of code from the Example Session:

```
<portType name="StudentImpl">
<operation name="getStudentDetails">
   <input message="impl:getStudentDetailsRequest" name="getStudentDetailsRequest" />
   <output message="impl:getStudentDetailsResponse" name="getStudentDetailsResponse" />
 </operation>
 </portType>
```

- The portType element defines a single operation, called getStudentDetails.
- The operation itself consists of a single input message getStudentDetailsRequest
- The operation itself consists of a single output message getStudentDetailsResponse

5)<operation>

Patterns of Operation:
WSDL supports four basic patterns of operation:

One-way :
The service receives a message. The operation therefore has a single *input* element.

syntax:

```
<definitions .... > <portType .... > *
     <operation name="nmtoken">
     <input name="nmtoken"? message="qname"/>
     </operation>
  </portType >
</definitions>
```

Request-response:
The service receives a message and sends a response. The operation therefore has one *input* element, followed by one *output* element. To encapsulate errors, an optional *fault* element can also be specified

syntax:

```
<definitions .... >
   <portType .... > *
     <operation name="nmtoken" parameterOrder="nmtokens">
       <input name="nmtoken"? message="qname"/>
       <output name="nmtoken"? message="qname"/>
```

```
      <fault name="nmtoken" message="qname"/>*
    </operation>
  </portType >
</definitions>
```

Solicit-response:

The service sends a message and receives a response. The operation therefore has one *output* element, followed by one *input* element. To encapsulate errors, an optional *fault* element can also be specified.

syntax:

```
<definitions .... >
  <portType .... > *
    <operation name="nmtoken" parameterOrder="nmtokens">
      <output name="nmtoken"? message="qname"/>
      <input name="nmtoken"? message="qname"/>
      <fault name="nmtoken" message="qname"/>*
    </operation>
  </portType >
</definitions>
```

Notification :

The service sends a message. The operation therefore has a single *output* element.

syntax:

```
<definitions .... >
  <portType .... > *
    <operation name="nmtoken">
      <output name="nmtoken"? message="qname"/>
      </operation>
  </portType >
</definitions>
```

6) <binding>

This element provides specific details on how a *portType* operation will actually be transmitted over the wire.

- The bindings can be made available via multiple transports, including HTTP GET, HTTP POST, or SOAP.
- The bindings provide concrete information on what protocol is being used to transfer *portType* operations.

The bindings provide information where the service is located.

- For SOAP protocol, the binding is **<soap:binding>**, and the transport is SOAP messages on top of HTTP protocol.
- You can specify multiple bindings for a single *portType*.

The binding element has two attributes - the name attribute and the type attribute.

```
<binding name="StudentImplSoapBinding" type="impl:StudentImpl">
  <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http" />
```

```
<operation name="getStudentDetails">
 <soap:operation soapAction="" />
<input name="getStudentDetailsRequest">
 <soap:body use="literal" />
 </input>
<output name="getStudentDetailsResponse">
 <soap:body use="literal" />
 </output>
 </operation>
 </binding>
```

The name attribute defines the name of the binding, and the type attribute points to the port for the binding, in this case the "impl: StudentImpl " port.

SOAP Binding:
WSDL 1.1 includes built-in extensions for SOAP 1.1. This enables you to specify SOAPspecific details, including SOAP headers, SOAP encoding styles, and the SOAPAction HTTP header. The SOAP extension elements include:

soap:binding
This element indicates that the binding will be made available via SOAP. The *style* attribute indicates the overall style of the SOAP message format. A style value of rpc specifies an RPC format.

The *transport* attribute indicates the transport of the SOAP messages. The value http://schemas.xmlsoap.org/soap/http indicates the SOAP HTTP transport, whereas http://schemas.xmlsoap.org/soap/smtp indicates the SOAP SMTP transport.

soap:operation
This element indicates the binding of a specific operation to a specific SOAP implementation. The soapAction attribute specifies that the SOAPAction HTTP header be used for identifying the service.

soap:body
This element enables you to specify the details of the input and output messages. In the case of HelloWorld. the body element specifies the SOAP encoding style and the namespace URN associated with the specified service.

Here is the piece of code from Example section:

```
<binding name="StudentImplSoapBinding" type="impl:StudentImpl">
 <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http" />
<operation name="getStudentDetails">
 <soap:operation soapAction="" />
<input name="getStudentDetailsRequest">
 <soap:body use="literal" />
 </input>
<output name="getStudentDetailsResponse">
 <soap:body use="literal" />
 </output>
 </operation>
 </binding>
```

7) <port>

This element defines an individual endpoint by specifying a single address for a binding.
Here is the grammer to specify a port:

```
<definitions .... >
  <service .... > *
    <port name="nmtoken" binding="qname"> *
    <-- extensibility element (1) -->
    </port>
  </service>
</definitions>
```

- The port element has two attributes - the name attribute and the binding attribute.
- The name attribute provides a unique name among all ports defined within in the enclosing WSDL document.
- The binding attribute refers to the binding using the linking rules defined by WSDL.
- Binding extensibility elements (1) are used to specify the address information for the port.
- A port MUST NOT specify more than one address.
- A port MUST NOT specify any binding information other than address information.

Here is the pice of code from Example session:

```
<service name="StudentImplService">
  <port binding="impl:StudentImplSoapBinding" name="StudentImpl">
    <soap:address location="http://localhost:8085/Provider_RPC/services/StudentImpl" />
  </port>
</service>
```

8)<service>

This element defines the ports supported by the Web service. For each of the supported protocols, there is one port element. The service element is a collection of ports.

- Web service clients can learn from the service element where to access the service, through which port to access the Web service, and how the communication messages are defined.
- The service element includes a documentation element to provide human-readable documentation.

Here is a piece of code from Example Session:

```
<service name="StudentImplService">
<documentation>WSDL File for StudentService</documentation>
 <port binding="impl:StudentImplSoapBinding" name="StudentImpl">
  <soap:address location="http://localhost:8085/Provider_RPC/services/StudentImpl" />
  </port>
  </service>
```

The binding attributes of port element associate the address of the service with a binding element defined in the Web service. In this example this is StudentImplSoapBinding

```
<binding name="StudentImplSoapBinding" type="impl:StudentImpl">
  <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http" />
<operation name="getStudentDetails">
  <soap:operation soapAction="" />
<input name="getStudentDetailsRequest">
  <soap:body use="literal" />
  </input>
<output name="getStudentDetailsResponse">
  <soap:body use="literal" />
  </output>
  </operation>
  </binding>
```

WSDL Elements mapping to Java

WSDL and XML construct	Java Construct
1) targetNamespace (eg:http://example.com)	package(name: com.example)
2) a) service(where is the service located?-The service element defines the address for invoking the specified service). b) port which contains in the service	a)Service Locator Class (Most commonly, this includes a URL for invoking the SOAP service). b)port accessor method in Service Locator Class.
3)binding (The binding element provides specific details on how a portType operation will actually be transmitted over the network). Style(attribute)=rpc(or)document. Use (attribute) =encoded (or) literal	STUB. Bindings can be made available via multiple transports, including HTTP GET, HTTP POST, or SOAP.
4)port Type(what operations will be supported?)	Interface(SEI)
5)operation	SEI Method
6)message	The SEI method signature typically is determined by the message
message-input	Parameters or arguments of method
message-output	Return types of method
message-fault	Throws Exceptions
7)xsd:complexType	User defined exceptions, Java beans, Arrays
xsd:simpleType	Java bean properties, Primitive types

Message Exchange Formats:

The WSDL file defines the format of the SOAP message(how to send request) that are transmitted

1. operation – <ns:display> //display is the method Name
2. message-input(parameter) name -- <a>

RPC/ENCODED	RPC/Literal	DOCUMENT/LITERAL	DOCUMENT/Wrapped
i)<soap:body>	<soap:body>	<soap:body>	<soap:body>
ii)<ns:display>	<ns:display>	--nothing will come here--	<ns:display_wrapper>
iii) <a xsi:type="xsd:string"> ABC	<a>ABC	<a>ABC	<a>ABC
iv)</ns:display>	</ns:display>	---nothing----	<ns:display_wrapper>
v)</soap:body>	</soap:body>	</soap:body>	</soap:body>
Advantage: 1)WSDL is self-explained. 2)The operation name is displayed.	Advantage: 1)WSDL is self-explained. 2)The operation name is displayed. 3)type info encoding is eliminated.	Advantage: 1) There is no type encoding. 2)Validate is very easy. 3)WS-I complaint	Advantage: 1)no type encoding 2)validate is easy
Dis-Advantage: 1)xsi:type is specified in SOAP is extra info. 2)Validate is difficult, few from XSD schema and few from WSDL 3)Not WS-I complaint.	Dis-Advantage: 1)Validate is difficult, few from XSD schema and few from WSDL	Dis-Advantage: 1)The operation name is not displayed. 2)WS-I allows one child of the soap:body	Dis-Advantage: 1)WSDL is more complicated.

JAX-RPC Handlers

JAX-RPC handlers allow you to intercept a SOAP message at various times during a service invocation. Handlers are similar to servlet filters.
Handlers can exist on both the client and the server side. If you use JAX-RPC on the client side, you can have a handler process a SOAP request message right before it goes on the network, and you can process the response message before it is returned to the client. Similarly, you can intercept an incoming SOAP request message on the server before invoking the service implementation, as well as the outgoing response.

Several handlers can be combined into what is called a "handler chain". Each handler processes the SOAP message, which is then passed on to the next handler in the chain. The exact sequence in which this happens is configurable.

To develop a JAX-RPC handler, you simply create a class that implements the javax.xml.rpc.handler.Handler interface. It has three methods to handle SOAP requests, responses and faults, respectively.

SOAP message handlers can be used to process messages to and from a Web service. There are two kinds of handlers: client and server.

- **Client side:** handlers can intercept messages sent from a client application, the "request", and the corresponding message returned from the service to the client, the "response".
- **Server side:** handlers can intercept messages received by a Web service, the "request", and the corresponding message returned by the service, the "response".

Common uses for handlers are:

- Logging
- Auditing
- Encryption/Decryption
- Authentication

JAX-RPC Handler Interfaces and Classes

javax.xml.rpc.handler Classes and Interfaces	Description
Handler	Main interface that is implemented when creating a handler. Contains methods to handle the SOAP request, response, and faults.
GenericHandler	Abstract class that implements the Handler interface. User should extend this class when creating a handler, rather than implement Handler directly. The GenericHandler class is a convenience abstract class that makes writing handlers easy. This class provides default implementations of the life cycle methods init and destroy and also different handle methods. A handler developer should only override methods that it needs to specialize as part of the derived handler implementation class.
HandlerChain	Interface that represents a list of handlers. An implementation class for the HandlerChain interface abstracts the policy and mechanism for the invocation of the registered handlers.
HandlerRegistry	Interface that provides support for the programmatic configuration of handlers in a HandlerRegistry.
HandlerInfo	Class that contains information about the handler in a handler chain. A HandlerInfo instance is passed in the Handler.init method to initialize a Handler instance.

MessageContext	Abstracts the message context processed by the handler. The MessageContext properties allow the handlers in a handler chain to share processing state.
soap.SOAPMessageContext	Sub-interface of the MessageContext interface used to get at or update the SOAP message.
javax.xml.soap.SOAPMessage	Object that contains the actual request or response SOAP message, including its header, body, and attachment.

Directly Manipulating the SOAP Request and Response Message Using SAAJ:

The javax.xml.soap.SOAPMessage abstract class is part of the SOAP With Attachments API for Java 1.1 (SAAJ) specification. You use the class to manipulate request and response SOAP messages when creating SOAP message handlers. This section describes the basic structure of a SOAPMessage object and some of the methods you can use to view and update a SOAP message.

A SOAPMessage object consists of a SOAPPart object (which contains the actual SOAP XML document) and zero or more attachments.

The SOAPPart Object

The SOAPPart object contains the XML SOAP document inside of a SOAPEnvelope object. You use this object to get the actual SOAP headers and body.

The following sample Java code shows how to retrieve the SOAP message from a MessageContext object, provided by the Handler class, and get at its parts:

```
SOAPMessage soapMessage = messageContext.getMessage();
SOAPPart soapPart = soapMessage.getSOAPPart();
SOAPEnvelope soapEnvelope = soapPart.getEnvelope();
SOAPBody soapBody = soapEnvelope.getBody();
SOAPHeader soapHeader = soapEnvelope.getHeader();
```

How to Write a JAX-RPC Handler:

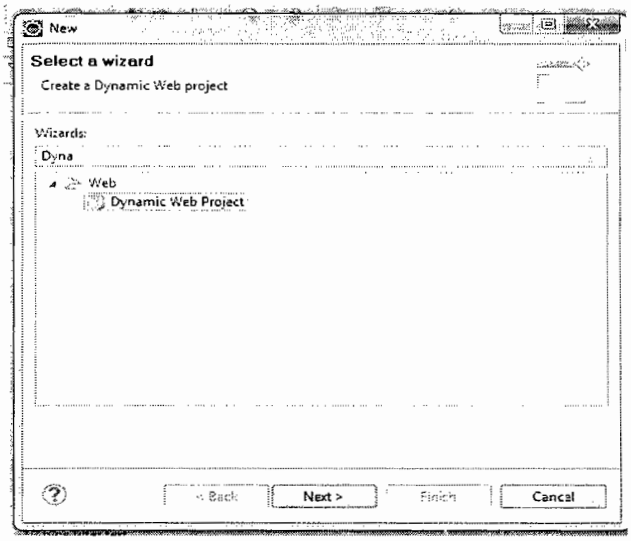
To write a JAX-RPC handler, implement the javax.xml.rpc.handler.Handler interface.

```
package javax.xml.rpc.handler;
public interface Handler{
   public boolean handleRequest(javax.xml.rpc.handler.MessageContext context):
   public boolean handleResponse(javax.xml.rpc.handler.MessageContext context):
   public boolean handleFault(javax.xml.rpc.handler.MessageContext context);
   public void destroy();
   public void init(javax.xml.rpc.handler.HandlerInfo config);
   public javax.xml.namespace.QName[] getHeaders():
}
```

Example:

PROVIDER:

Step 1: Create a Dynamic Web Project (Provider_JAXPRC_Handler)



Step 2: Write an SEI interface and Implementation class.

IStudent.java

```
package com.nit.service;

import java.rmi.Remote;

import java.rmi.RemoteException;

public interface IStudent extends Remote{

        public String getStudentDetails(Integer sno)throws RemoteException;

}
```

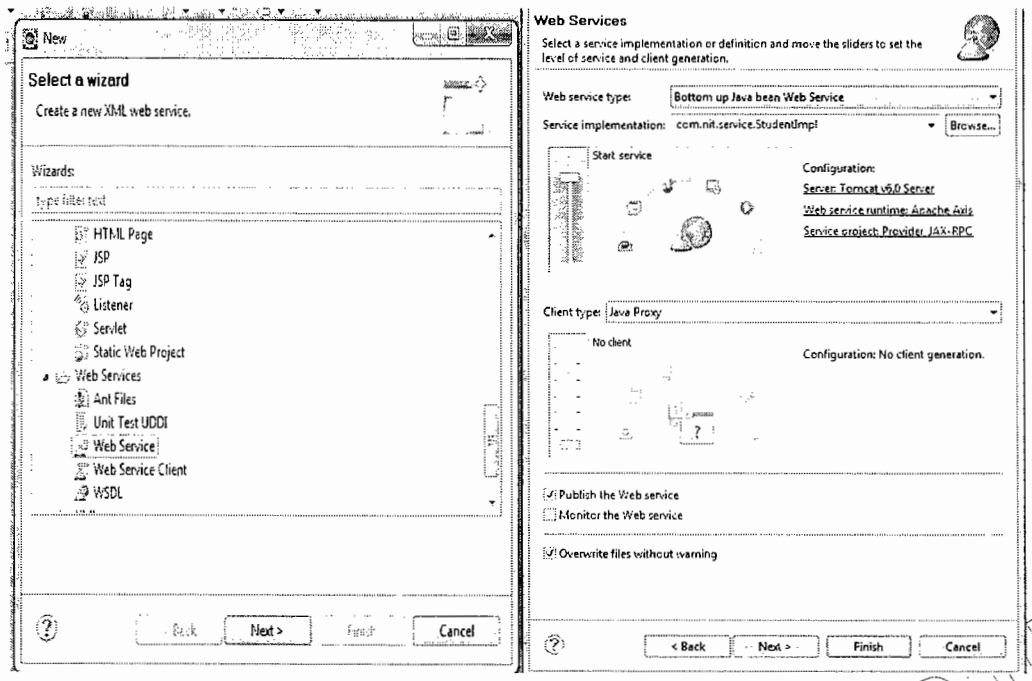
StudentImpl.java

```
package com.nit.service;

import java.io.IOException;

import java.rmi.RemoteException;

import java.util.Iterator;

import javax.xml.rpc.ServiceException;

import javax.xml.rpc.handler.MessageContext;

import javax.xml.rpc.handler.soap.SOAPMessageContext;

import javax.xml.rpc.server.ServiceLifecycle;

import javax.xml.rpc.server.ServletEndpointContext;

import javax.xml.soap.SOAPEnvelope;

import javax.xml.soap.SOAPException;

import javax.xml.soap.SOAPHeader;

import javax.xml.soap.SOAPMessage;

import javax.xml.soap.SOAPPart;

import org.w3c.dom.NodeList;
```

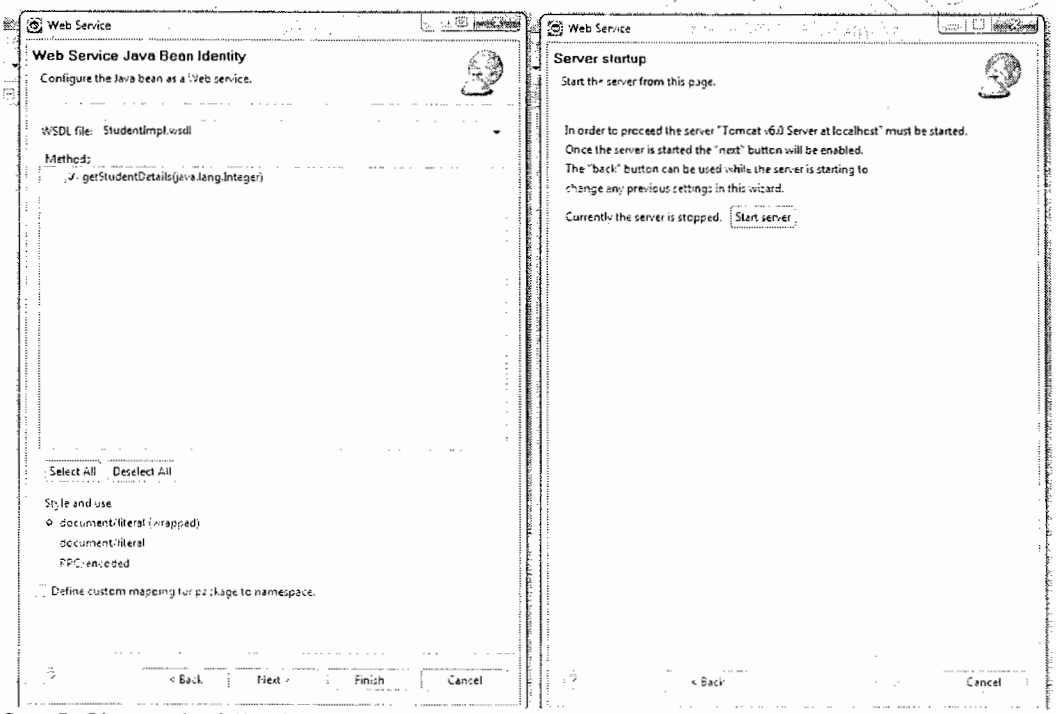
```
public class StudentImpl implements IStudent, ServiceLifecycle {
	ServletEndpointContext ctx;
	@Override
	public String getStudentDetails(Integer sno) throws RemoteException {
		boolean userValid = isValidUser();
		if (userValid) {
			String name = null;
			if (sno == 001) {
				name = "Raju";
			} else {
				name = "Rani";
			}
			return name;
		} else {
			return "User is Not Valid";
		}
	}
	@Override
	public void destroy() {
		// TODO Auto-generated method stub
	}
	@Override
	public void init(Object arg0) throws ServiceException {
		// TODO Auto-generated method stub
		System.out.println("init is called");

		if (javax.xml.rpc.server.ServletEndpointContext.class.isInstance(arg0)) {
			System.out.println("if condition is called");
			ctx = (ServletEndpointContext) arg0;
		}
	}
	public boolean isValidUser() {
		System.out.println("object:" + ctx);
		boolean valid = false;
		if (ctx != null) {
			MessageContext mc = (MessageContext) ctx.getMessageContext();
			SOAPMessageContext soapMessageContext = (SOAPMessageContext) mc;
```

```
                try {
                        SOAPMessage message = soapMessageContext.getMessage();
                        System.out.print("req data:");
                        message.writeTo(System.out);
                        SOAPPart soapPart = message.getSOAPPart();
                        SOAPEnvelope soapEnvelope = soapPart.getEnvelope();
                        SOAPHeader header = soapEnvelope.getHeader();

                        NodeList userIdNode = header.getElementsByTagName("username");
                        String userId = userIdNode.item(0).getChildNodes().item(0)
                                        .getNodeValue();
                        System.out.println("username:" + userId);
                        NodeList passwordNode = header.getElementsByTagName("password");
                        String password = passwordNode.item(0).getChildNodes().item(0)
                                        .getNodeValue();
                        System.out.println("password:" + password);

                        if (userId.equals("Raju") && password.equals("Secret")) {
                                valid = true;
                        }

                } catch (SOAPException e) {
                        e.printStackTrace();
                } catch (IOException e) {
                        e.printStackTrace();
                }

                // process SOAP header as shown in the message handler
        }
        return valid;

    }
}
```

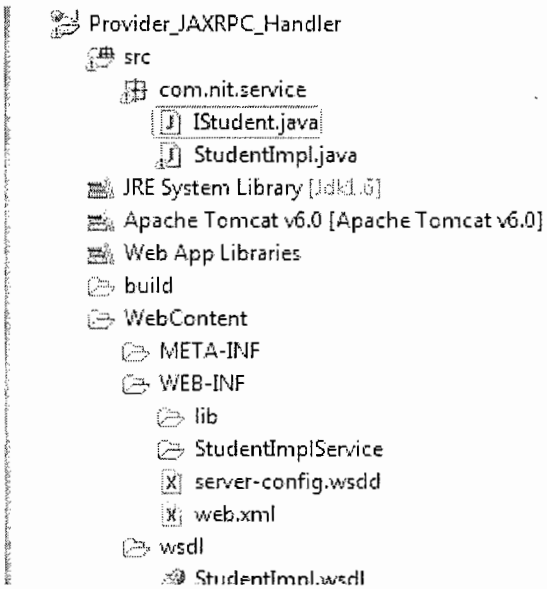
Step 2: Right click on project -->New-->Other-->Web Service
Step 3: Browse an Implementation class and select Bottom up java bean Web Service as shown below.
Step 4: click on check box. Publish the Web Service



Step 5: Select the method which exposes as service.
Step 6: Start Server



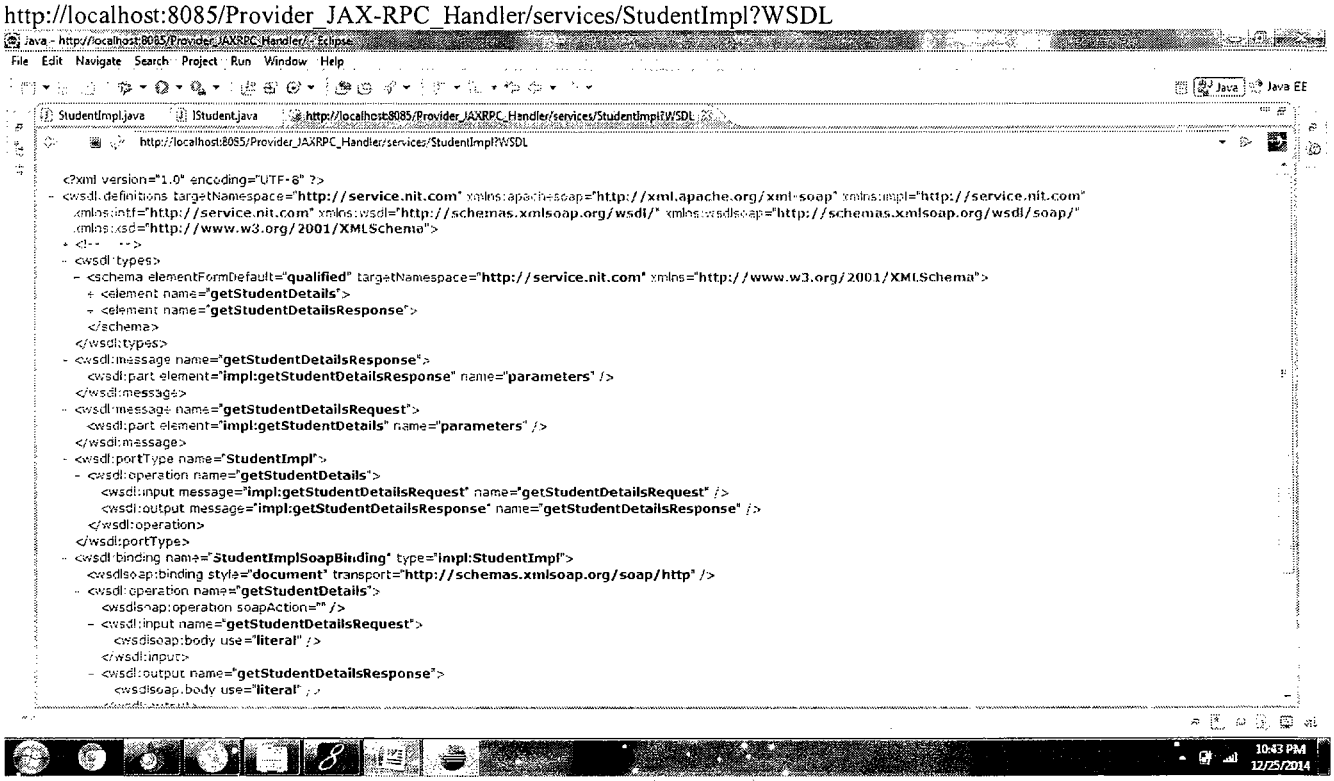
Step 7: Observe the following.
a) WSDL file in WSDL folder
b) server-config.wsdd
c) web.xml



web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://java.sun.com/xml/ns/javaee"
xmlns:web="http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd" xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd" id="WebApp_ID" version="2.5">
 <display-name>Provider_JAX-RPC</display-name>
  <servlet>
   <display-name>Apache-Axis Servlet</display-name>
   <servlet-name>AxisServlet</servlet-name>
   <servlet-class>org.apache.axis.transport.http.AxisServlet</servlet-class>
  </servlet>
  <servlet-mapping>
   <servlet-name>AxisServlet</servlet-name>
   <url-pattern>/servlet/AxisServlet</url-pattern>
  </servlet-mapping>
  <servlet-mapping>
   <servlet-name>AxisServlet</servlet-name>
   <url-pattern>*.jws</url-pattern>
  </servlet-mapping>
  <servlet-mapping>
   <servlet-name>AxisServlet</servlet-name>
   <url-pattern>/services/*</url-pattern>
  </servlet-mapping>
  <servlet>
   <display-name>Axis Admin Servlet</display-name>
   <servlet-name>AdminServlet</servlet-name>
   <servlet-class>org.apache.axis.transport.http.AdminServlet</servlet-class>
   <load-on-startup>100</load-on-startup>
  </servlet>
  <servlet-mapping>
   <servlet-name>AdminServlet</servlet-name>
   <url-pattern>/servlet/AdminServlet</url-pattern>
  </servlet-mapping>
</web-app>
```

Step 8: Browse the following URL.
http://localhost:8085/Provider_JAX-RPC_Handler/services/StudentImpl?WSDL

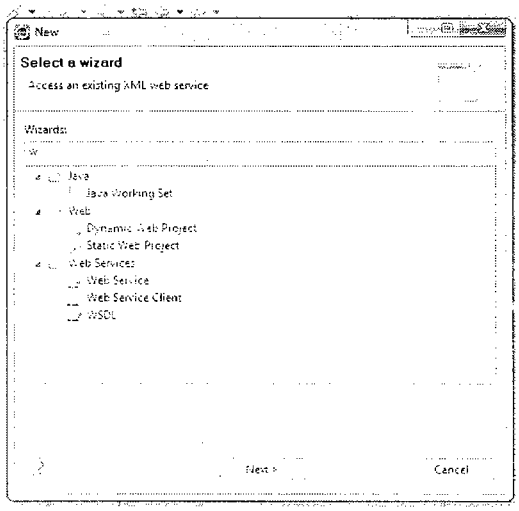


CONSUMER:

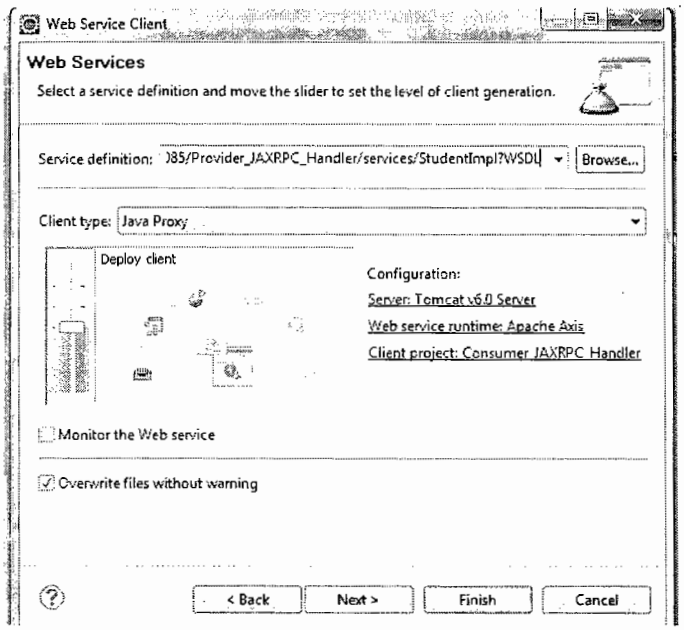
Consumer with Client-Side Handler:

Step 1: Create a java project (Consumer_JAX-RPC_Handler)

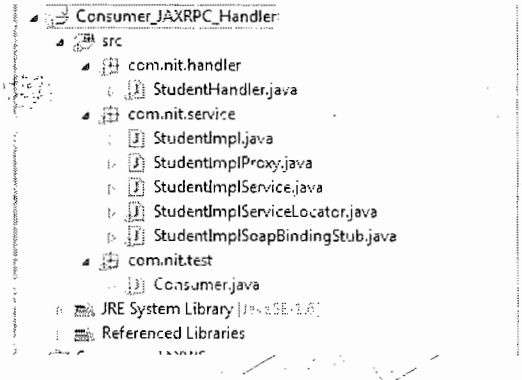
Step 2: Right click -on project -->New-->Other-->Web Service Client.



Step 3: Browse the URL of WSDL file as shown below screen.



Step 4: Observe the client-side artifacts in project.



Step 5: write an Handler Class.

StudentHandler.**java**

```
package com.nit.handler;

import java.io.ByteArrayOutputStream;
import java.io.IOException;
import java.util.Map;

import javax.xml.namespace.QName;
import javax.xml.rpc.handler.Handler;
import javax.xml.rpc.handler.HandlerInfo;
import javax.xml.rpc.handler.MessageContext;
import javax.xml.rpc.handler.soap.SOAPMessageContext;
import javax.xml.soap.Name;
import javax.xml.soap.SOAPElement;
import javax.xml.soap.SOAPEnvelope;
import javax.xml.soap.SOAPException;
import javax.xml.soap.SOAPFactory;
import javax.xml.soap.SOAPHeader;
import javax.xml.soap.SOAPHeaderElement;
```

```
import javax.xml.soap.SOAPMessage;

public class StudentHandler implements Handler {
        private QName headers[];
        private String username = "";
        private String password = "";
        private Map configMap = null;
        public StudentHandler() {
                System.out.println("New StudentHandler instance created.");
        }

        public void init(HandlerInfo config) {
                System.out.println("Inside INIT of the StudentHandler.");
                headers = config.getHeaders();
                configMap = config.getHandlerConfig();
                if (configMap != null && configMap.size() > 0) {
                        System.out.println("Header: " + config.getHandlerConfig());
                        username = (String) configMap.get("username");
                        password = (String) configMap.get("password");
                }
        }

        public QName[] getHeaders() {
                return headers;
        }
        public boolean handleRequest(MessageContext context) {
                System.out.println("Inside handleRequest");
                SOAPMessageContext soapMsgCtx = null;
                try {
                        soapMsgCtx = (SOAPMessageContext) context;
                        SOAPMessage soapMessage = ((SOAPMessageContext) context)
                                        .getMessage();
                        SOAPHeader header = soapMessage.getSOAPHeader();
                        SOAPEnvelope envelope = soapMessage.getSOAPPart().getEnvelope();
                        if (header == null) {
                                header = envelope.addHeader();
                        }
                        System.out.println("username: " + username);
                        System.out.println("password: " + password);
                Name qNameUserCredentials = envelope.createName(
                                        "Header", "soapenv", null);

                        SOAPHeaderElement userCredentials = header
                                        .addHeaderElement(qNameUserCredentials);

                        Name qNameUsername = envelope.createName(
                                        "username");

                        SOAPHeaderElement usernameElement = header.addHeaderElement(qNameUsername);
                        usernameElement.addTextNode(this.username);
                        Name qNamePassword = envelope.createName(
                                        "password");

                        SOAPHeaderElement passwordElement = header.addHeaderElement(qNamePassword);
                        passwordElement.addTextNode(this.password);

                        userCredentials.addChildElement(usernameElement);
                        userCredentials.addChildElement(passwordElement);
```

```
                System.out.println("Control header added");
                logSOAPMessage(soapMsgCtx);
            } catch (SOAPException e) {
                logSOAPMessage(soapMsgCtx);
                e.printStackTrace();
            } catch (Exception e) {
                logSOAPMessage(soapMsgCtx);
                e.printStackTrace();
            }
            return true;
        }

        public boolean handleResponse(MessageContext context) {
            SOAPMessageContext messageContext = (SOAPMessageContext) context;
            logSOAPMessage(messageContext);
            return true;
        }

        public boolean handleFault(MessageContext context) {
            SOAPMessageContext messageContext = (SOAPMessageContext) context;
            logSOAPMessage(messageContext);
            return true;
        }

        private void logSOAPMessage(SOAPMessageContext messageContext) {
            ByteArrayOutputStream out = new ByteArrayOutputStream();
            try {
                messageContext.getMessage().writeTo(out);
                System.out.println(new String(out.toByteArray()));
            } catch (SOAPException ex) {
                ex.printStackTrace();
            } catch (IOException ioex) {
                ioex.printStackTrace();
            }
        }

        public void destroy() {
        }


}
```

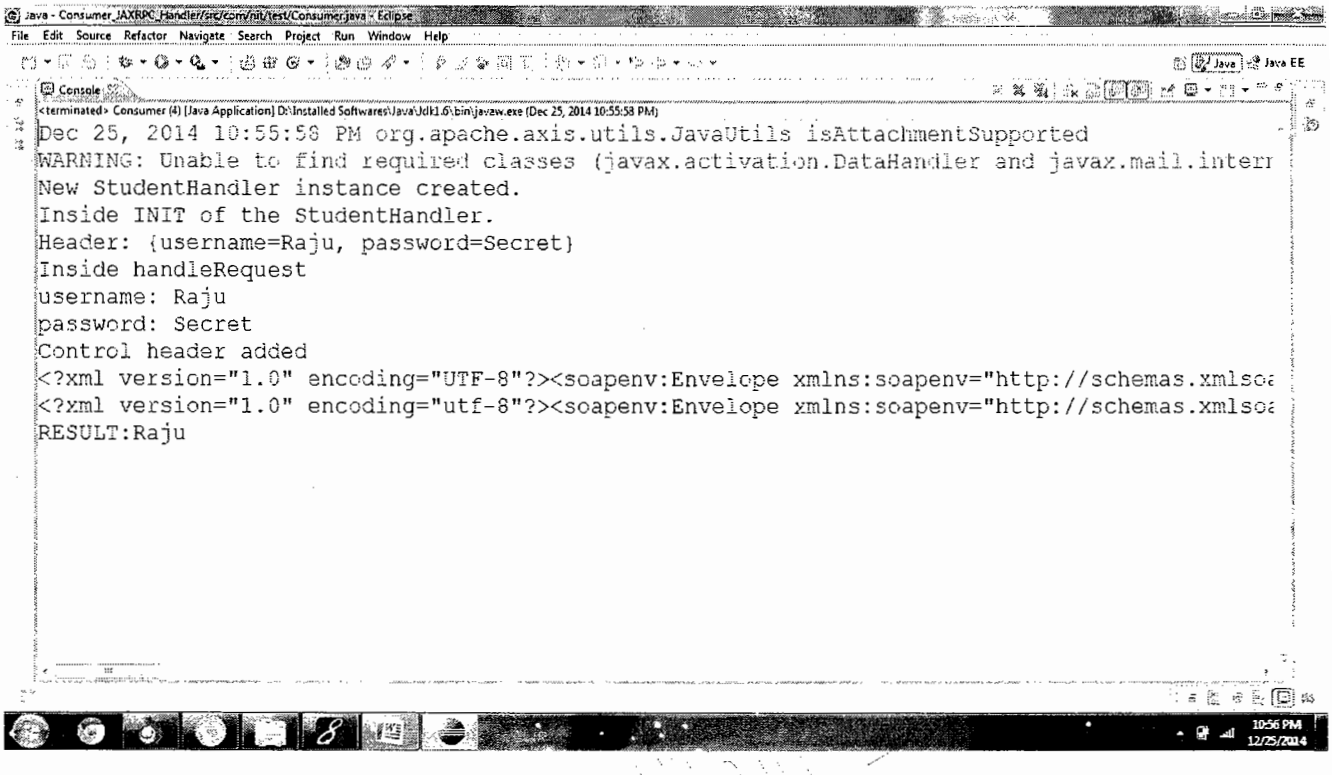
Step 6: write an client class to test the application.

Consumer.java

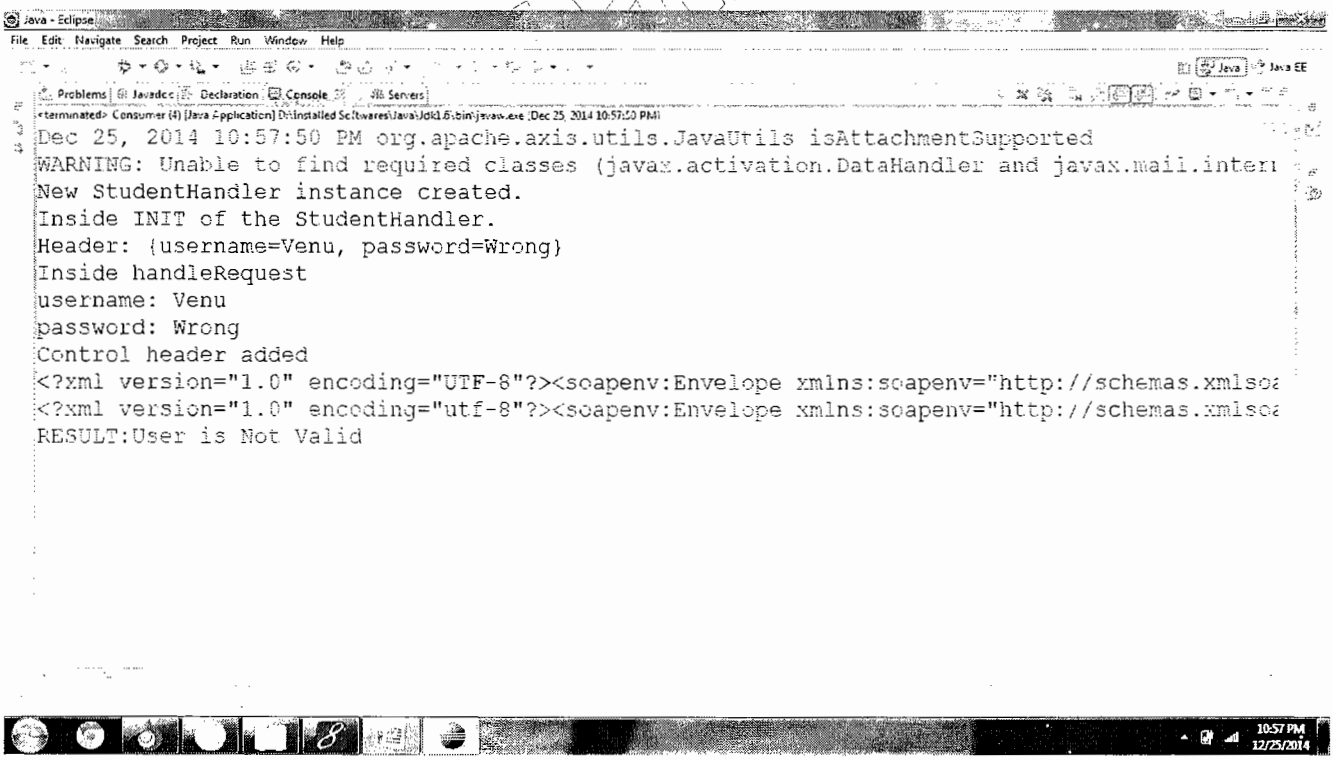
```
package com.nit.test;

import java.net.MalformedURLException;
import java.net.URL;
import java.rmi.RemoteException;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

import javax.xml.namespace.QName;
import javax.xml.rpc.Service;
import javax.xml.rpc.ServiceException;
import javax.xml.rpc.ServiceFactory;
import javax.xml.rpc.handler.HandlerInfo;
```

```
import javax.xml.rpc.handler.HandlerRegistry;

import com.nit.handler.StudentHandler;
import com.nit.service.StudentImpl;

public class Consumer {
        public static void main(String[] args) throws RemoteException,
                        ServiceException, MalformedURLException {
                QName serviceName = new QName(
                                "http://service.nit.com",
                                "StudentImplService");

                URL wsdlLocation = new URL(
                                "http://localhost:8085/Provider_JAXRPC_Handler/services/StudentImpl?WSDL");

                // Service
                ServiceFactory factory = ServiceFactory.newInstance();
                Service service = factory.createService(wsdlLocation, serviceName);

                HandlerRegistry hr = service.getHandlerRegistry();

                QName portName = new QName(
                                "http://service.nit.com",
                                "StudentImpl");

                List handlerChain = hr.getHandlerChain(portName);
                Map config = new HashMap();

                config.put("username", "Raju");
                config.put("password", "Secret");

                handlerChain.add(new HandlerInfo(StudentHandler.class, config, null));


                StudentImpl studentImpl = (StudentImpl) service.getPort(portName,
                                StudentImpl.class);
                String name = studentImpl.getStudentDetails(001);
                System.out.println("RESULT:" + name);      }

}
```

Result with Valid Username and Password:



Result with In-Valid Username and Password:



JAX-WS

JAX-WS

Java API for XML Web Services (JAX-WS) , is a set of APIs for creating web services in XML format (SOAP). JAX-WS provides many annotation to simplify the development and deployment for both web service clients and web service providers (endpoints).

In JAX-WS, a web service operation invocation is represented by an XML-based protocol such as SOAP. The SOAP specification defines the envelope structure, encoding rules, and conventions for representing web service invocations and responses. These calls and responses are transmitted as SOAP messages (XML files) over HTTP.

→It stands for Java API for XML-based web services

→It's a successor of JAX-RPC

→JAX-WS is a library and runtime environment that uses JAX-B internally to prepare WSDL from java classes.

SEI(Service Endpoint Interface)

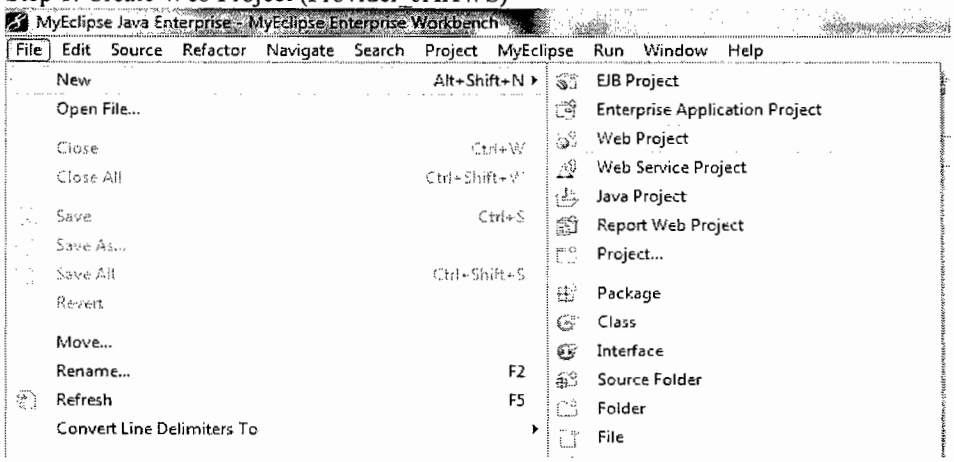
JAX-WS endpoints must follow these requirements.

- The implementing class must be annotated with javax.jws.WebServicc i.e @WebService
- The implementing class may explicitly reference an SEI through the endpointInterface element of the @WebService annotation but is not required to do so.
- If no endpointInterface is specified in @WebService, an SEI is implicitly defined for the implementing class.
- The business methods of the implementing class must be public and must not be declared static or final.
- Business methods that are exposed to web service clients must be annotated with javax.jws.WebMethod. i.e @WebMethod.
- Business methods that are exposed to web service clients must have JAXB-compatible parameters and return types.
- The implementing class must not be declared final and must not be abstract.
- The implementing class must have a default public constructor.
- The implementing class must not define the finalize method.

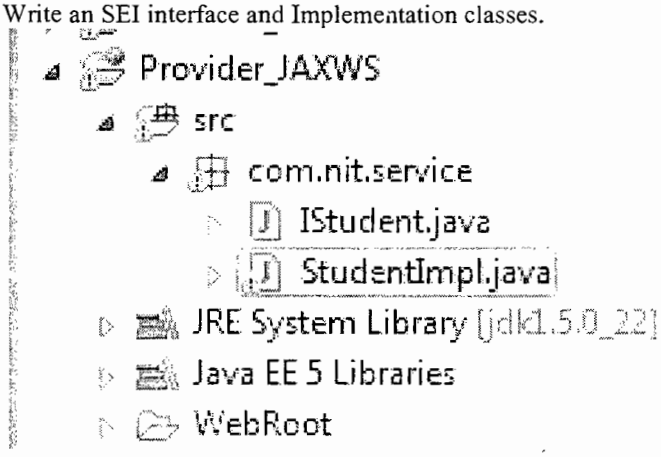
JAX-WS Metro Implementation (Using MyEclipse IDE)

PROVIDER:

Step 1: Create Web Project (Provider_JAXWS)



Step 2: Write an SEI interface and Implementation classes.



IStudent.java

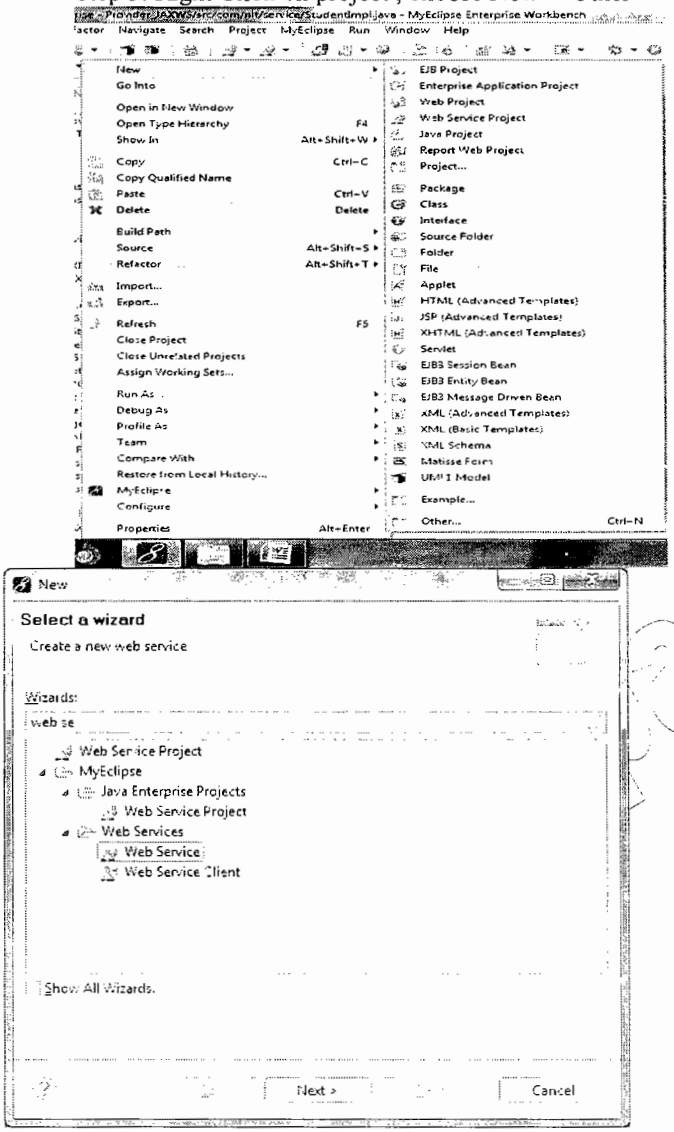
```
package com.nit.service;
import javax.jws.WebMethod;
import javax.jws.WebService;
import javax.xml.ws.WebServiceException;
@WebService
public interface IStudent {
        @WebMethod
        public String getStudentDetails(Integer sno)throws WebServiceException;
}
```

StudentImpl.java

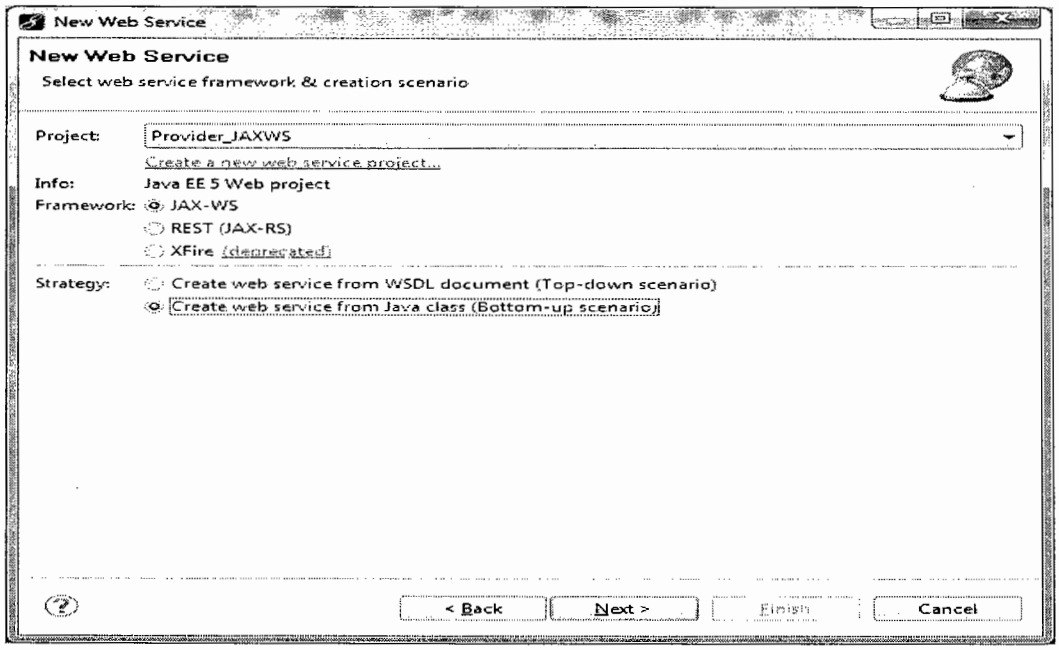
```
package com.nit.service;
import javax.jws.WebMethod;
import javax.jws.WebService;
import javax.xml.ws.WebServiceException;
@WebService
public class StudentImpl implements IStudent {
        @WebMethod
        public String getStudentDetails(Integer sno) throws WebServiceException {
                String name = null;
```

```
                if (sno == 001) {
                        name = "Raju";
                } else if (sno == 002) {
                        name = "Ravi";
                } else {
                        name = "Mantri";
                }
                return name;
        }
}
```

Step 3: Right Click on project , choose New-->Other-->Web Service

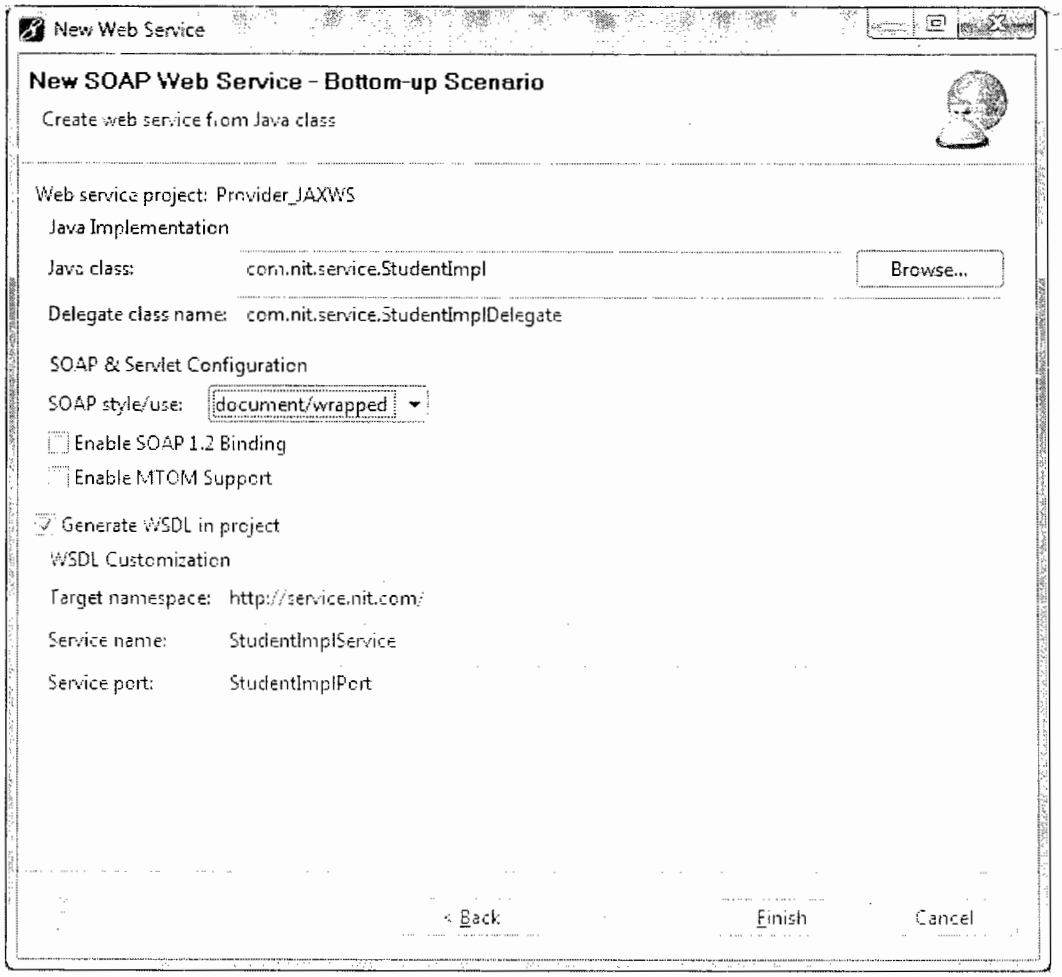


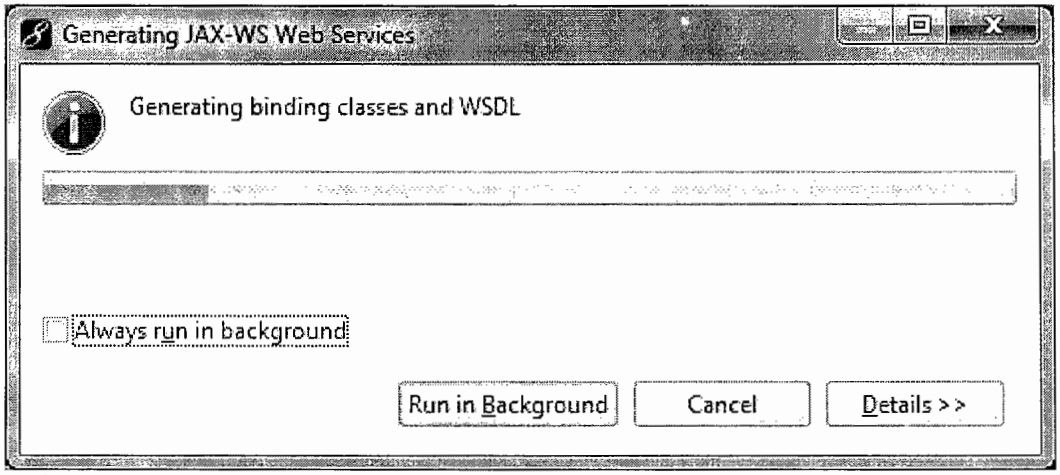
Step 4: Choose JAX-WS and Bottom-Up Scenario as shown below.



Step 4: Follow the below steps:

a) Choose our Implementation class
b) Select message-exchange format (ex. document/wrapped)
c) Click on Check box-Generate WSDL in project
d) Observe Target Name space, Service Name, Service Port
e) If you want to change the names change it)
f) Click on Finish, It will generate Provider Artifacts as shown below.



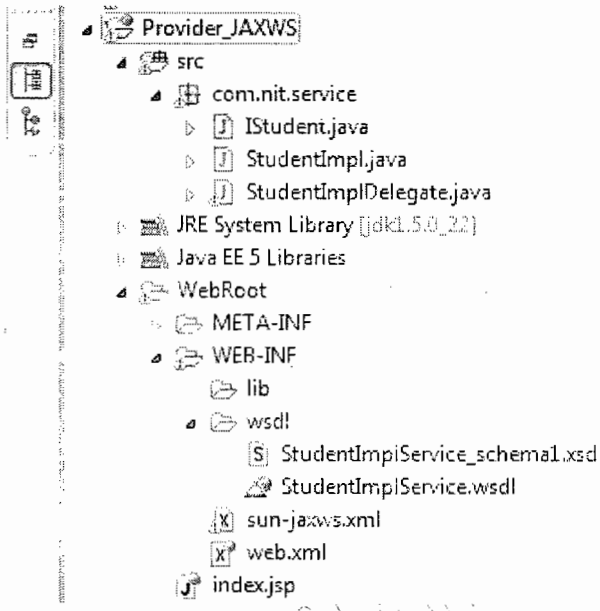


Step 5: Observe the project structure.

a) It is modifying the web.xml as shown below.

a. Generating WSDL in the project.

b) Generated sun-jaxws.xml



web.xml

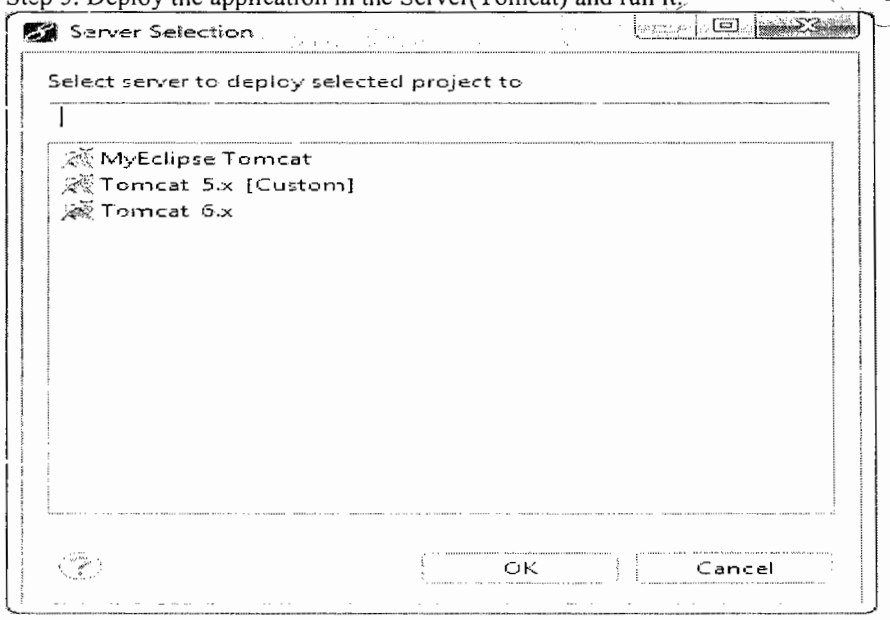
```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.5"
        xmlns="http://java.sun.com/xml/ns/javaee"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
        http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd">
  <servlet>
        <description>JAX-WS endpoint - StudentImplService</description>
        <display-name>StudentImplService</display-name>
        <servlet-name>StudentImplService</servlet-name>
        <servlet-class>
                com.sun.xml.ws.transport.http.servlet.WSServlet
        </servlet-class>
        <load-on-startup>1</load-on-startup>
  </servlet>
```

```
<servlet-mapping>
        <servlet-name>StudentImplService</servlet-name>
        <url-pattern>/StudentImplPort</url-pattern>
</servlet-mapping>
<welcome-file-list>
  <welcome-file>index.jsp</welcome-file>
</welcome-file-list>
<listener>
        <listener-class>
                com.sun.xml.ws.transport.http.servlet.WSServletContextListener
        </listener-class>
</listener></web-app>
```

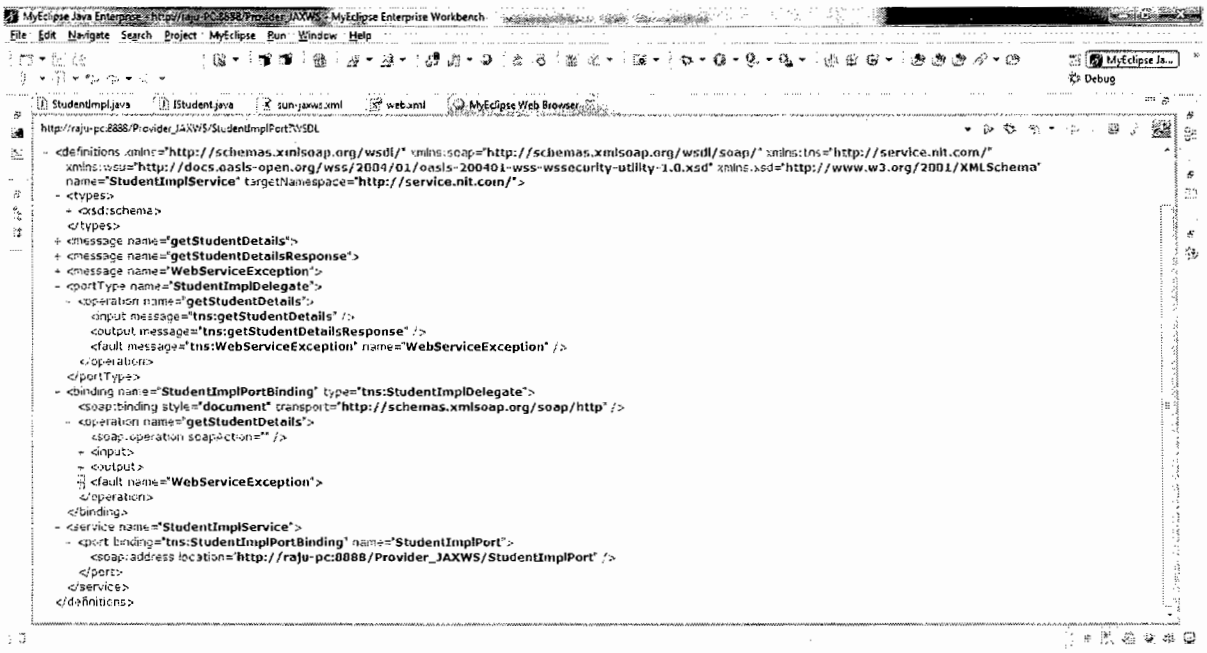
sun-jaxws.xml

```
<?xml version = "1.0"?>
<endpoints version="2.0"
        xmlns="http://java.sun.com/xml/ns/jax-ws/ri/runtime">
        <endpoint name="StudentImplPort"
                implementation="com.nit.service.StudentImplDelegate"
                url-pattern="/StudentImplPort">
        </endpoint></endpoints>
```

Step 5: Deploy the application in the Server(Tomcat) and run it.

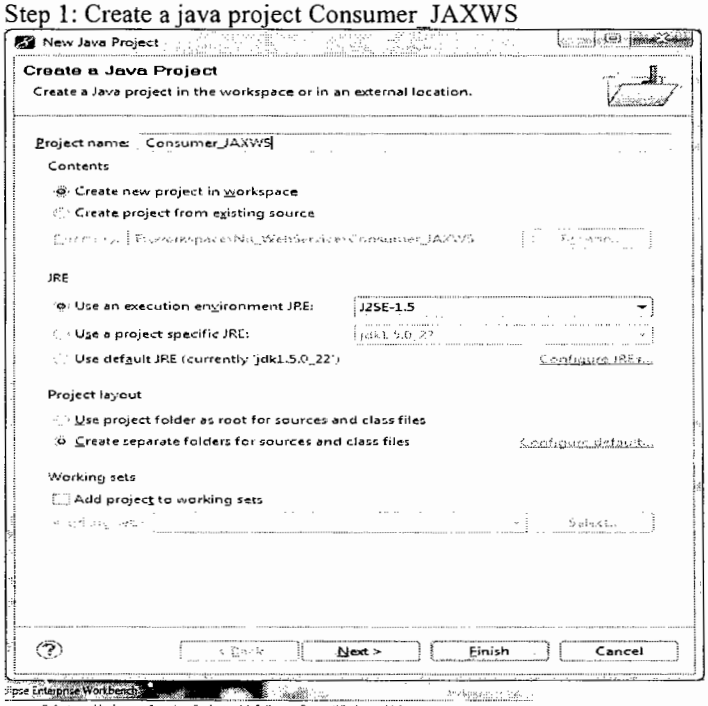


Step 6: Run the following URL and check the WSDL file.
http://localhost:8888/Provider_JAXWS/StudentImplPort?WSDL

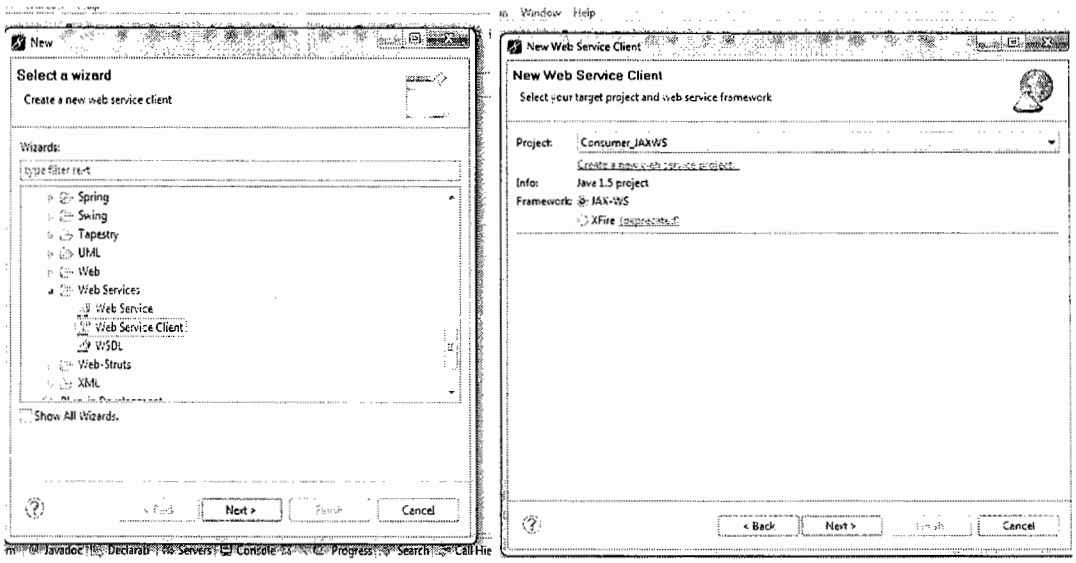


CONSUMER:

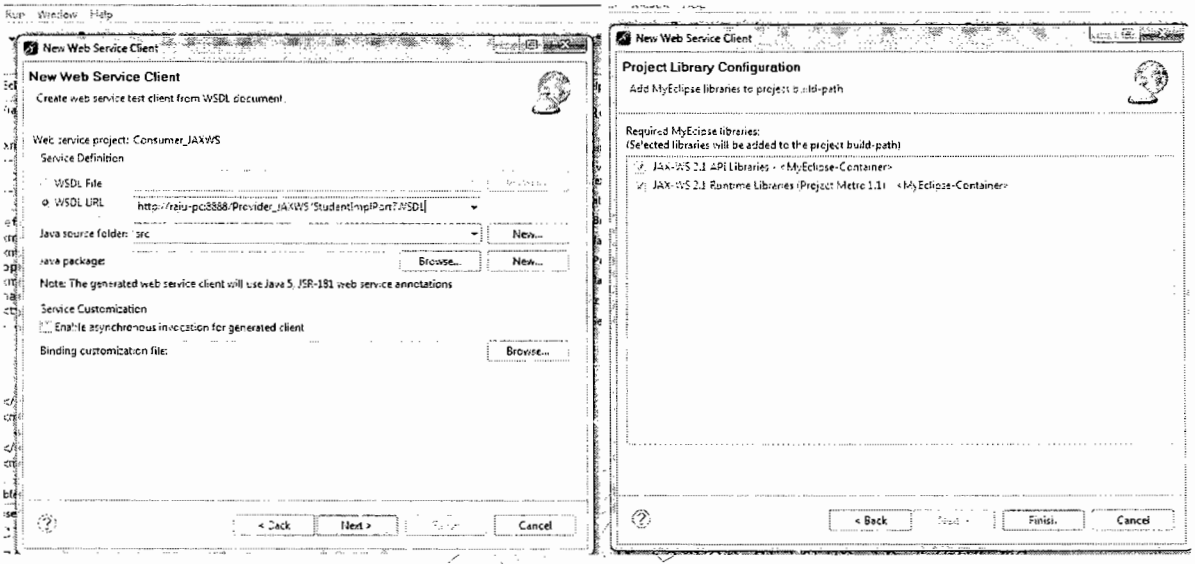
Step 1: Create a java project Consumer_JAXWS



Step 2: Right click on project New-->Other-->Web Service Client



Step 3: Enter the URL of WSDL (which is running above Provider)
http://localhost:8888/Provider_JAXWS/StudentImplPort?WSDL



Step 4: Observe the client-side artifacts as shown in below.

Step 5: Write a Client class to test the application using client-side artifacts.

```
package com.nit.test;
import com.nit.service.StudentImplDelegate;
import com.nit.service.StudentImplService;
import com.nit.service.WebServiceException_Exception;
public class Client {
        public static void main(String[] args) throws WebServiceException_Exception {
                StudentImplService service = new StudentImplService();
                StudentImplDelegate portType = service.getStudentImplPort();
                System.out.println(portType.getStudentDetails(001));
        }

}
```

.NET Frame work

JAVA Provider- .Net Consumer:

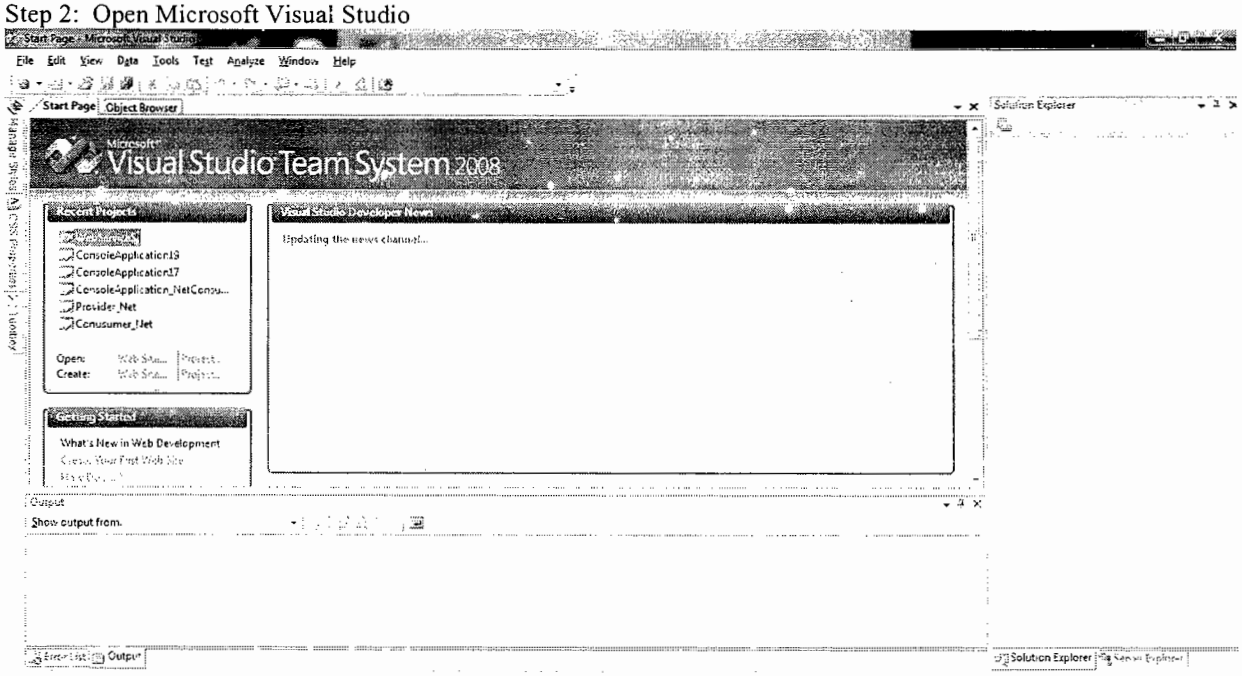
PROVIDER:

- Use the above Provider_JAXWS as Java Provider.
- Deploy the project into Server
- Run the following URL for WSDL file

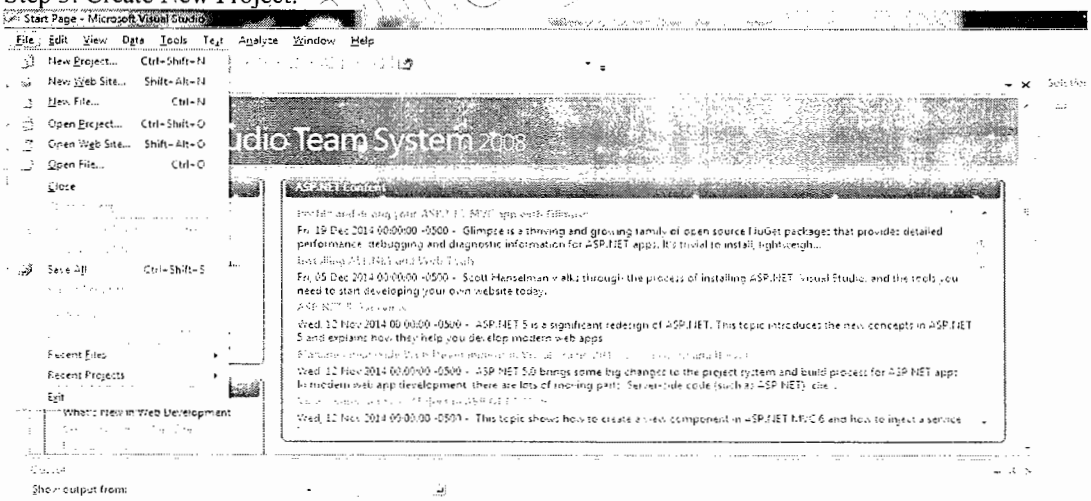
http://raju-pc:8888/Provider_JAXWS/StudentImplPort?WSDL

CONSUMER:

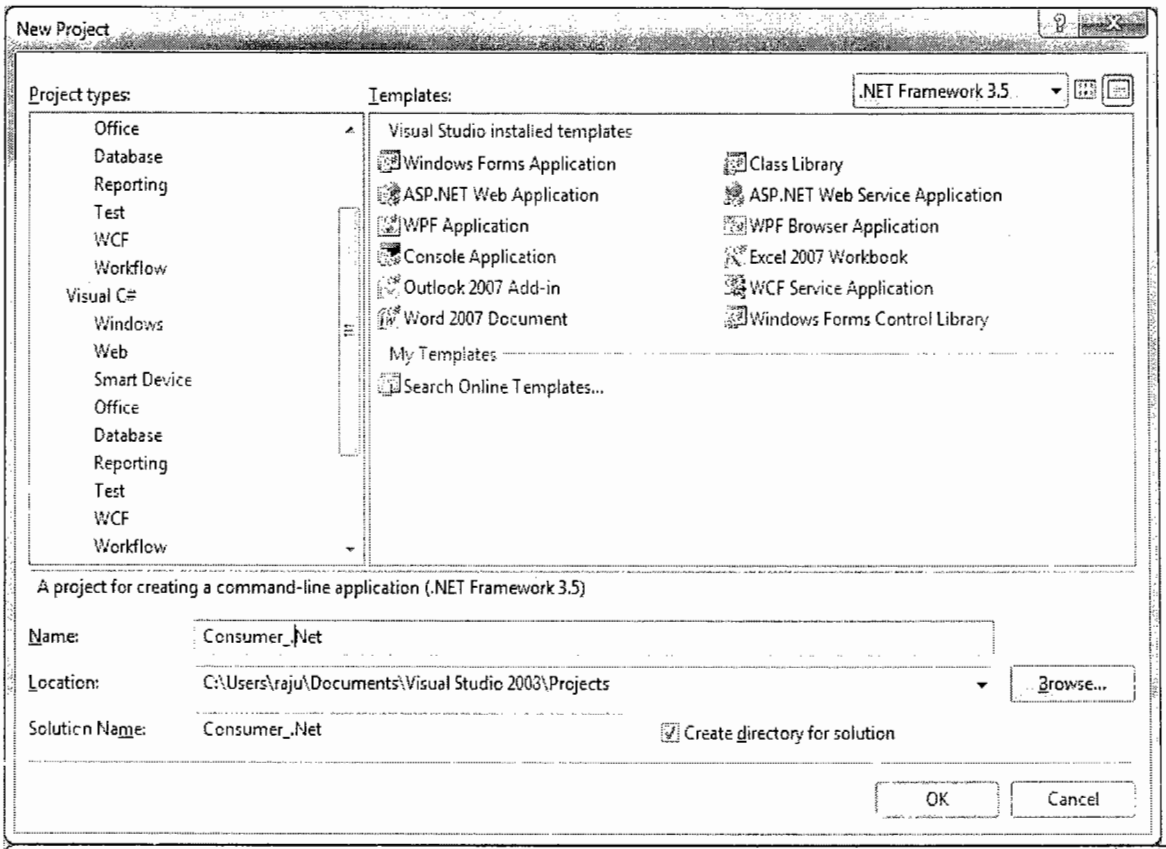
Step 1: Download and Install Microsoft Visual Studio.
Step 2: Open Microsoft Visual Studio

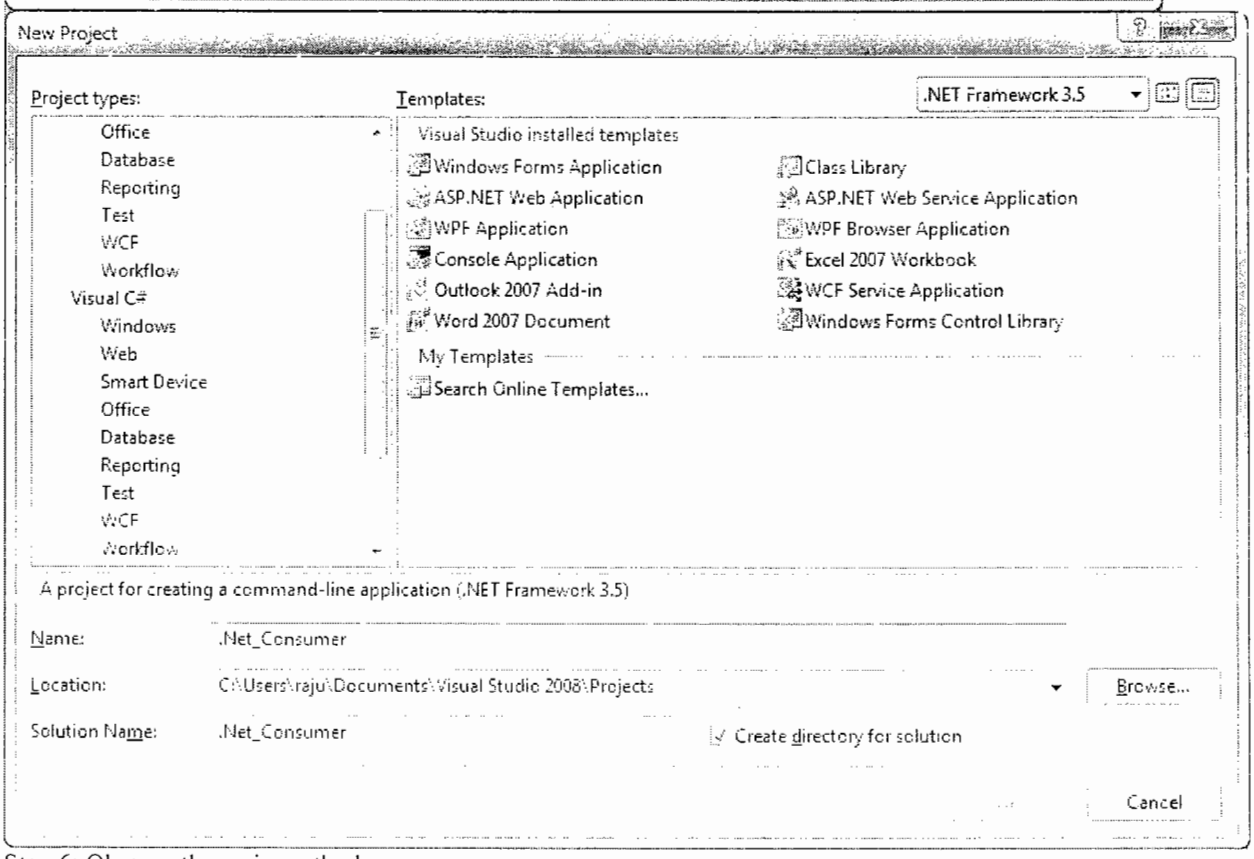


Step 3: Create New Project.

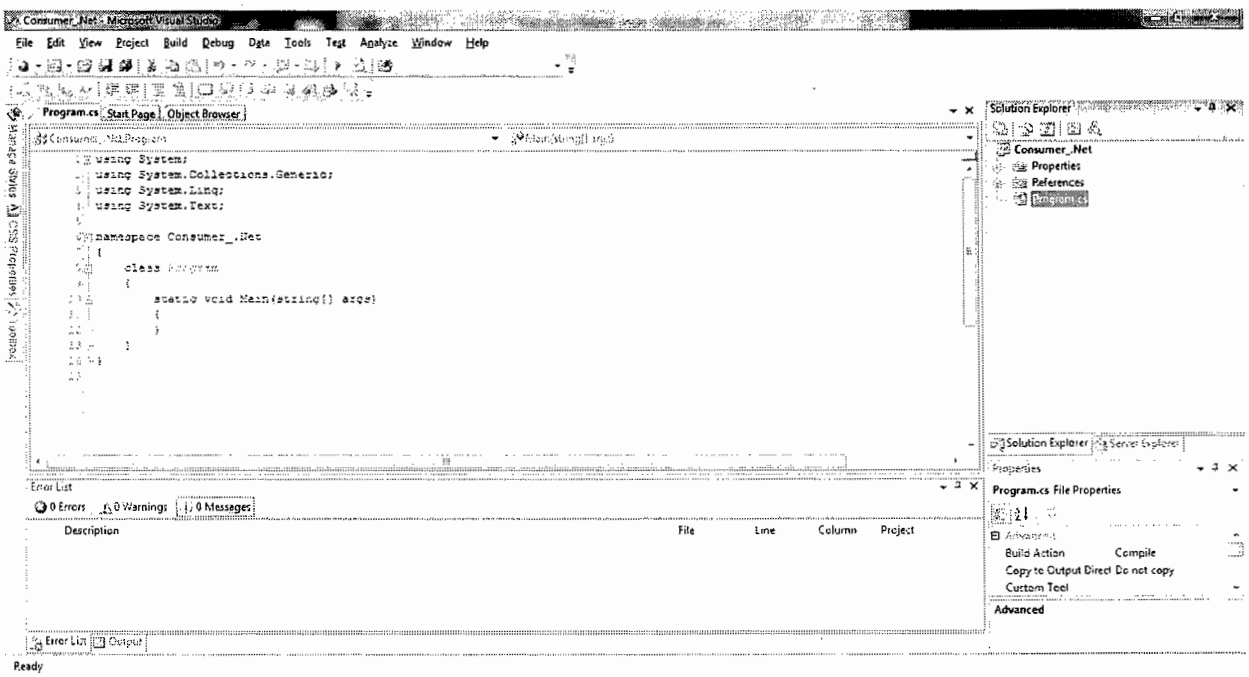


Step 4: Create C#- Console Application as shown below.
Step 5: Enter the Project name as (Consumer_.Net)

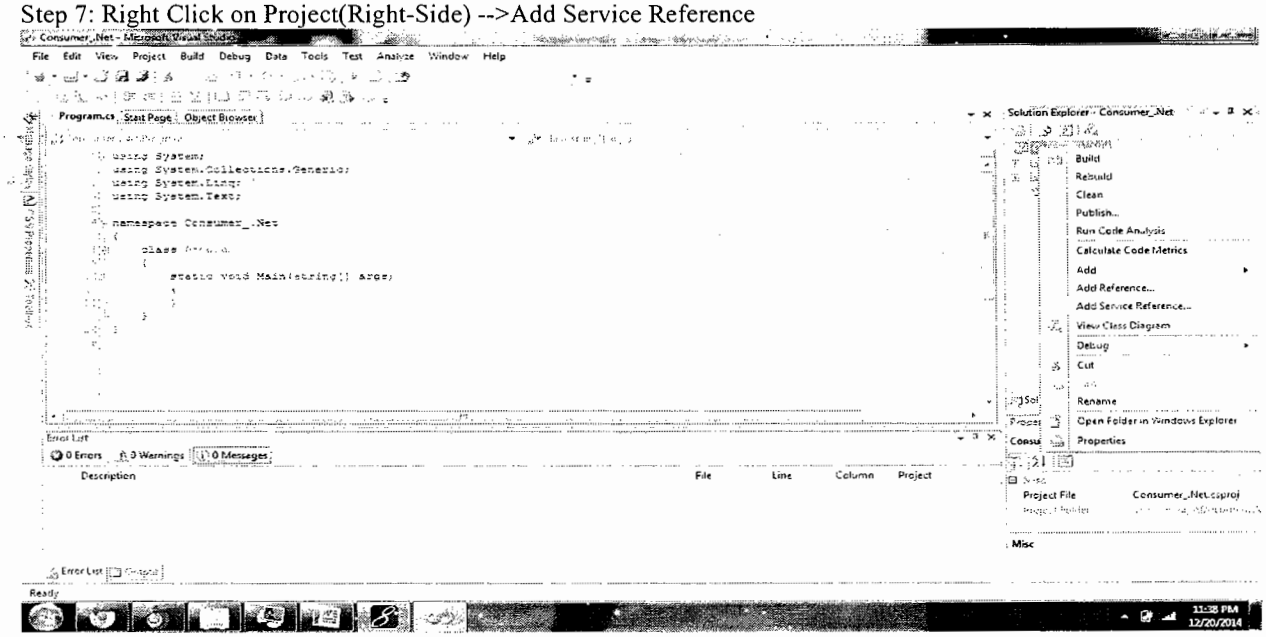




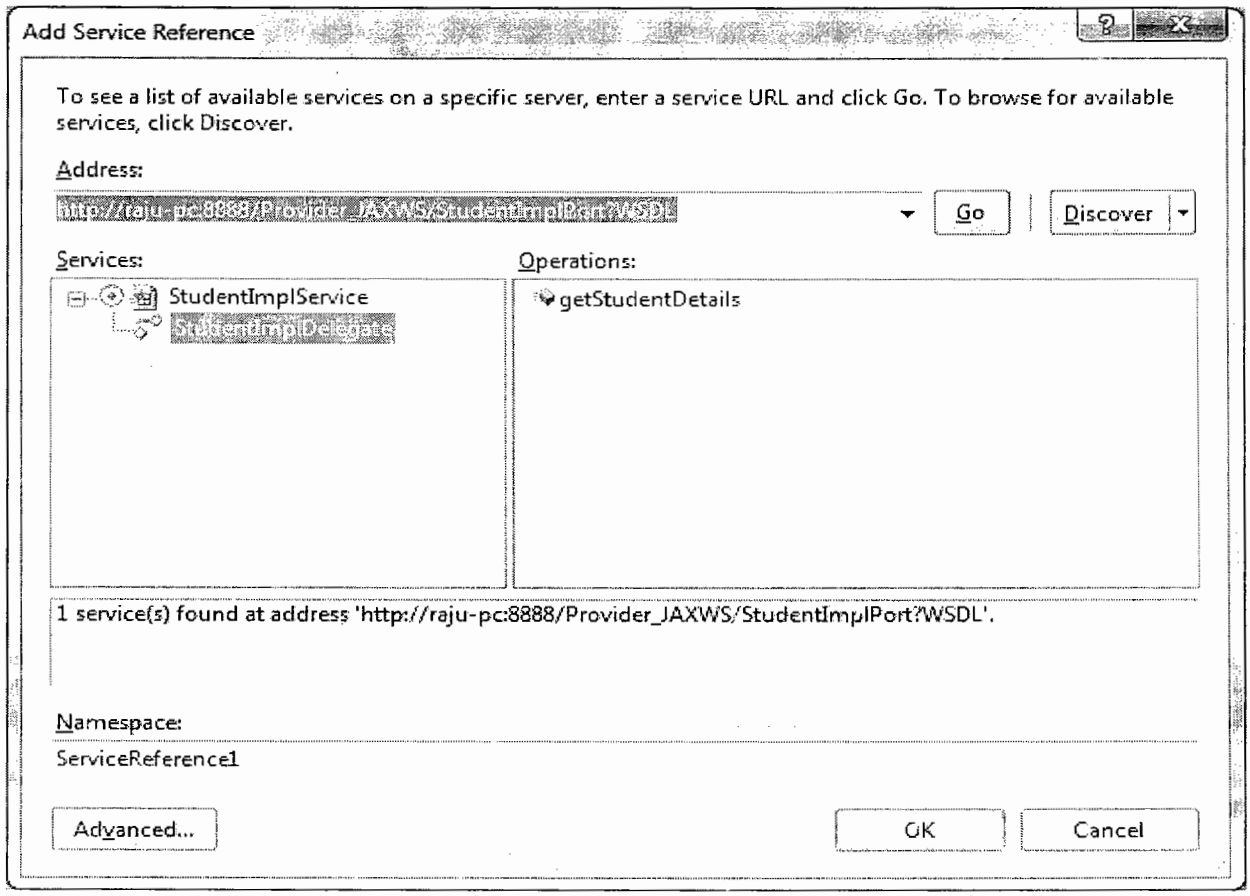
Step 6: Observe the main method.



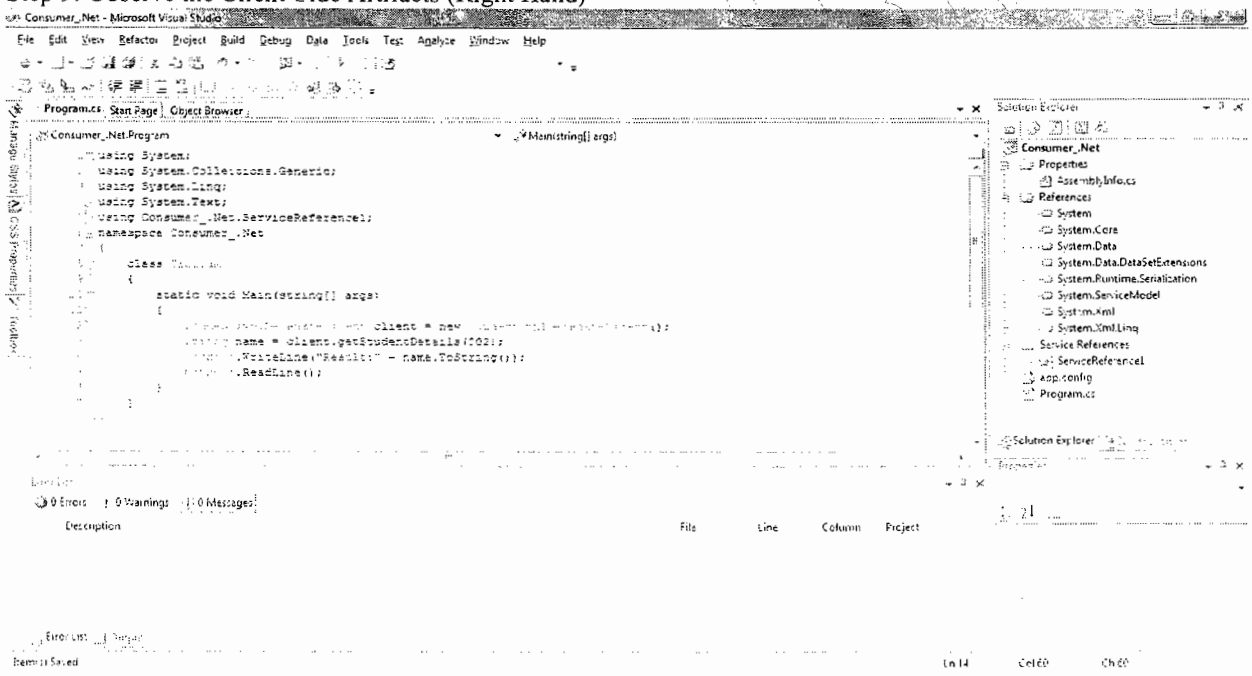
Step 7: Right Click on Project(Right-Side) -->Add Service Reference



Step 8: Enter the URL of WSDL, Click on OK

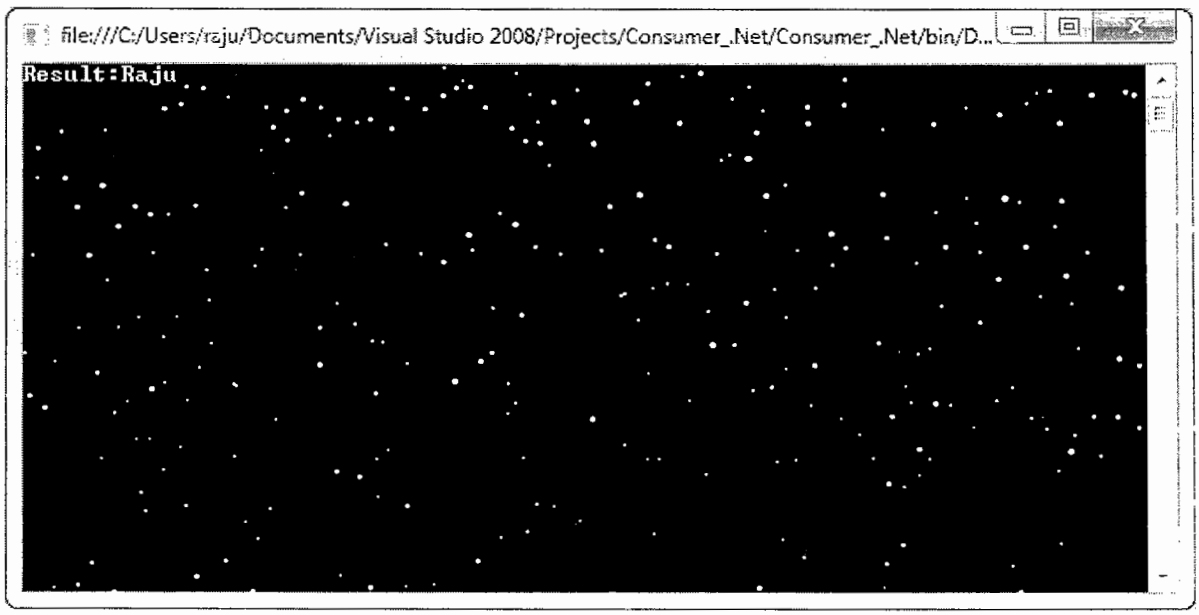


Step 9: Observe the Client-Side Artifacts (Right Hand)



Step 10: write the following code and test it.

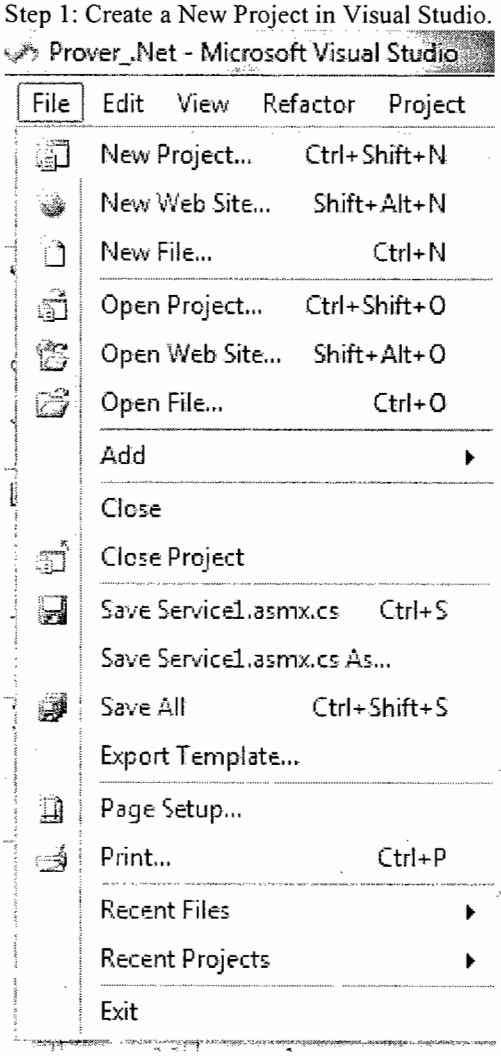
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using Consumer_.Net.ServiceReference1;
namespace Consumer_.Net
{
  class Program
  {
    static void Main(string[] args)
    {
      StudentImplDelegateClient client = new StudentImplDelegateClient();
      String name = client.getStudentDetails(001);
      Console.WriteLine("Result:" + name.ToString());
      Console.ReadLine();
    }
  }
}
```



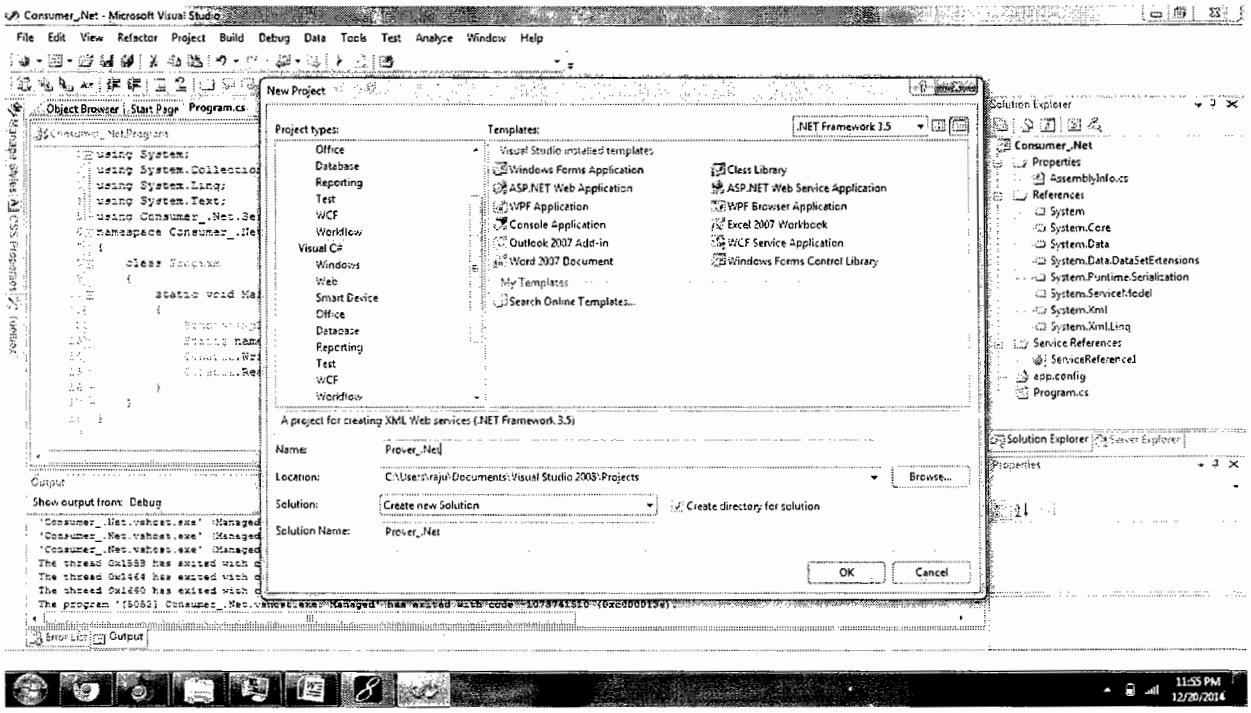
.Net Provider- .Java Consumer:

PROVIDER:

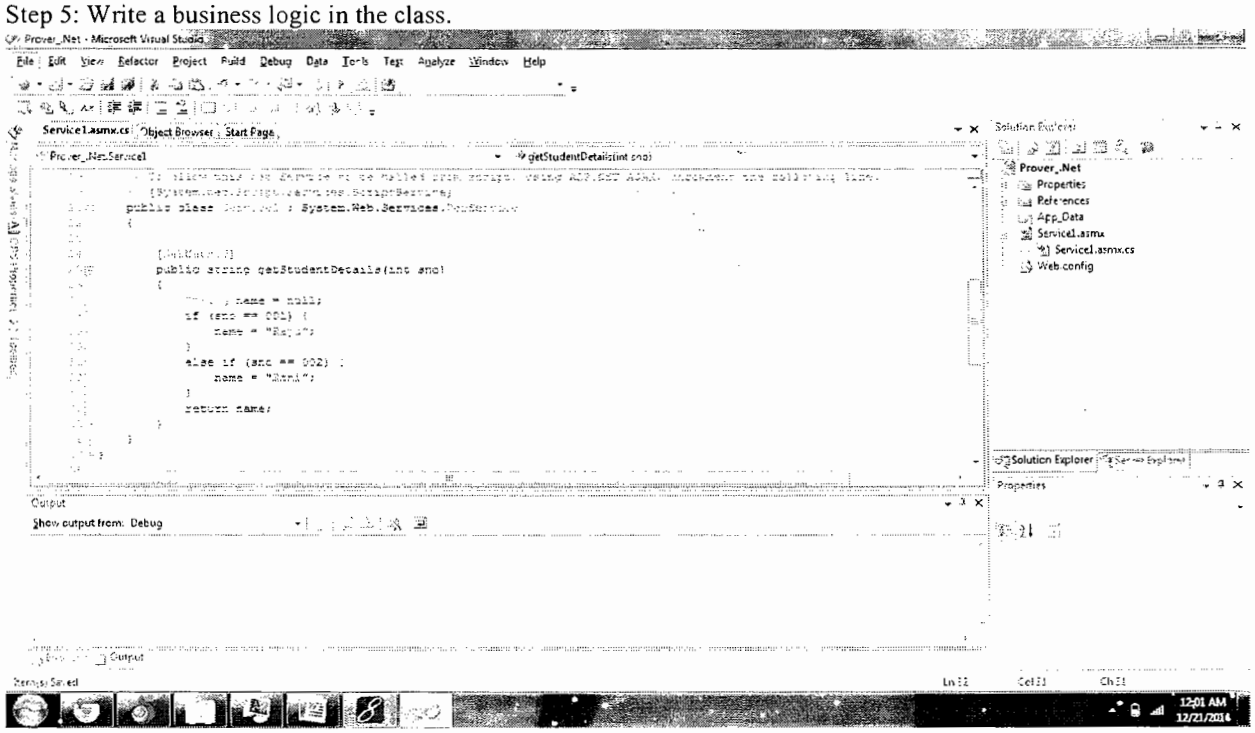
Step 1: Create a New Project in Visual Studio.



Step 2: Chose ASP.Net Web Service Application.
Step 3: Enter the Project Name as Provider_.Net.
Step 4: Click on OK Button.



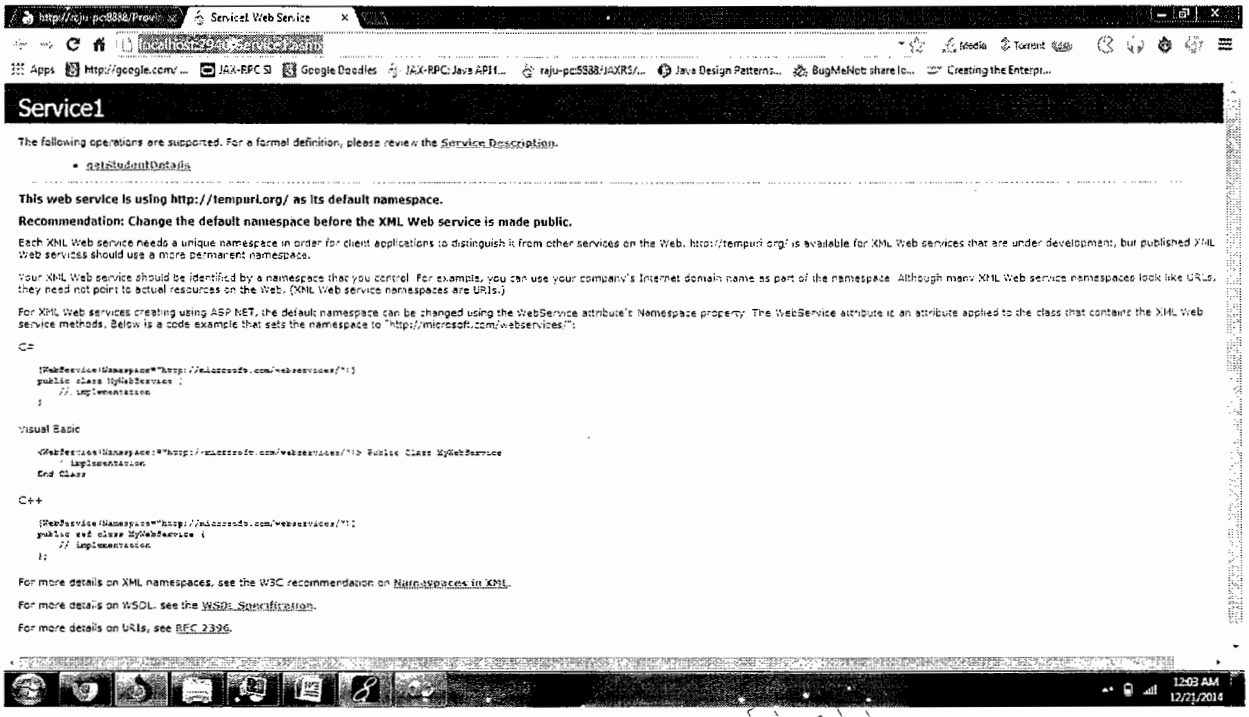
Step 5: Write a business logic in the class.



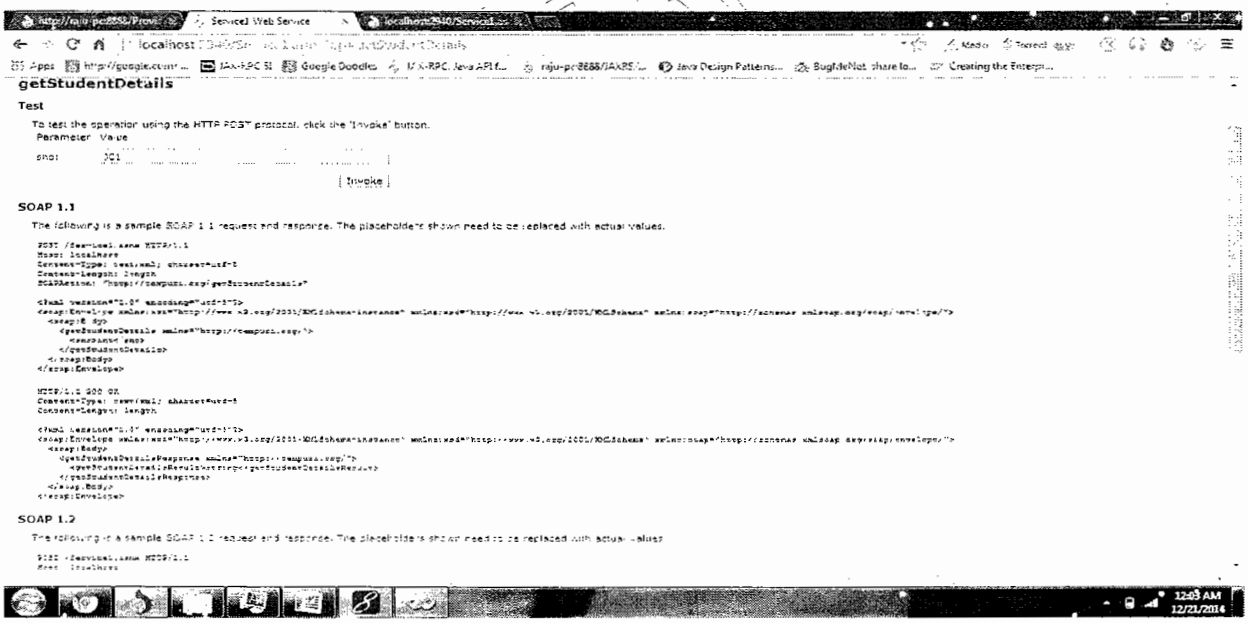

```
using System;
using System.Collections;
using System.ComponentModel;
using System.Data;
using System.Linq;
using System.Web;
using System.Web.Services;
using System.Web.Services.Protocols;
```

```
using System.Xml.Linq;

namespace Prover_.Net
{
  /// <summary>
  /// Summary description for Service1
  /// </summary>
  [WebService(Namespace = "http://tempuri.org/")]
  [WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1_1)]
  [ToolboxItem(false)]
  // To allow this Web Service to be called from script, using ASP.NET AJAX, uncomment the following line.
  // [System.Web.Script.Services.ScriptService]
  public class Service1 : System.Web.Services.WebService
  {

    [WebMethod]
    public string getStudentDetails(int sno)
    {
      String name = null;
      if (sno == 001) {
        name = "Raju";
      }
      else if (sno == 002) {
        name = "Rani";
      }
      return name;
    }
  }
}
```

Step 6: Click on Run Button(Green Color)



Step 7: Click on getStudentDetails link.
Step 8: Pass input and Click on Invoke Button.
Step 9: Observe SOAP Request and SOAP Response.

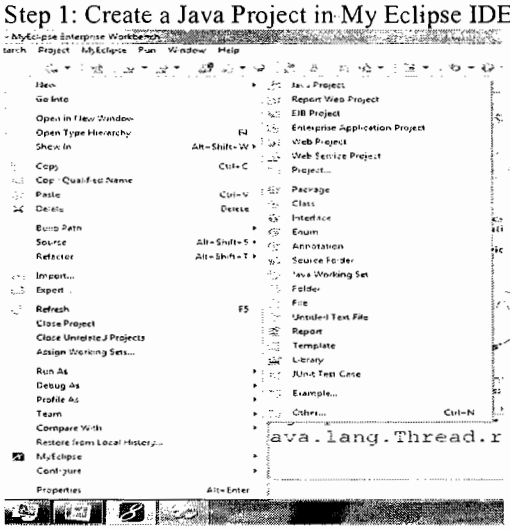


Step 10: Hit the following URL to Check WSDL
http://localhost:2940/Service1.asmx?WSDL

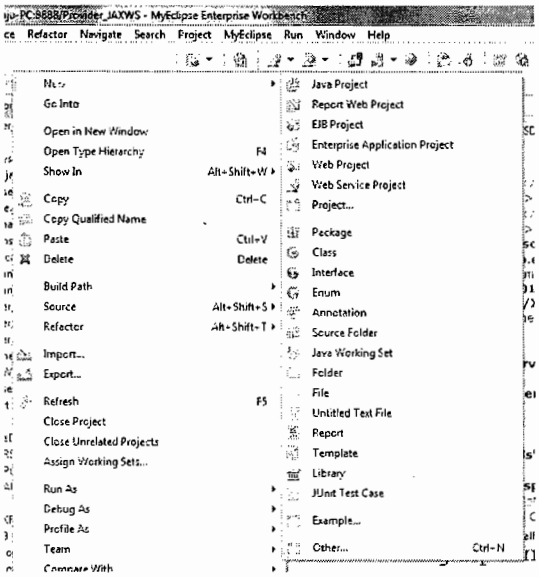
Step 11: Test the Service with SOAP UI Tool.

CONSUMER:

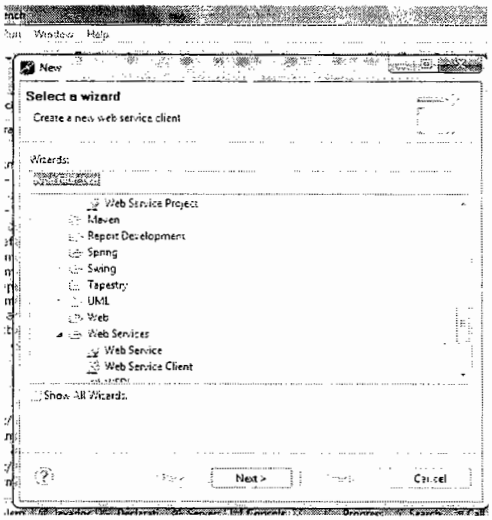
Step 1: Create a Java Project in My Eclipse IDE



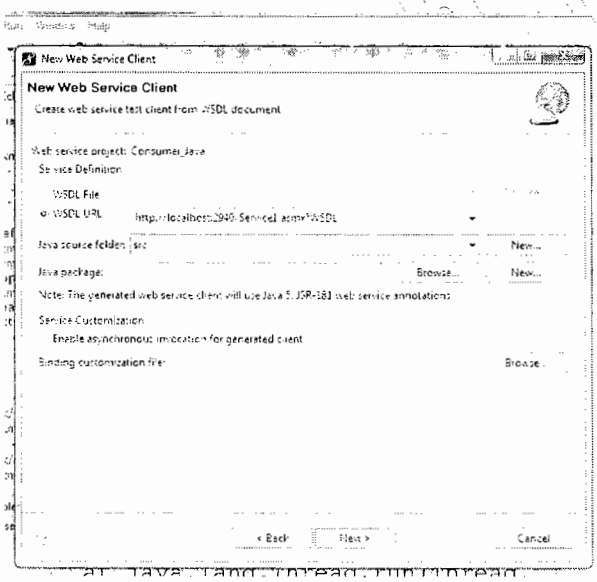
Step 2: Enter the Project Name as (Consumer_Java)
Step 3: Right Click on Project -->New--> Other



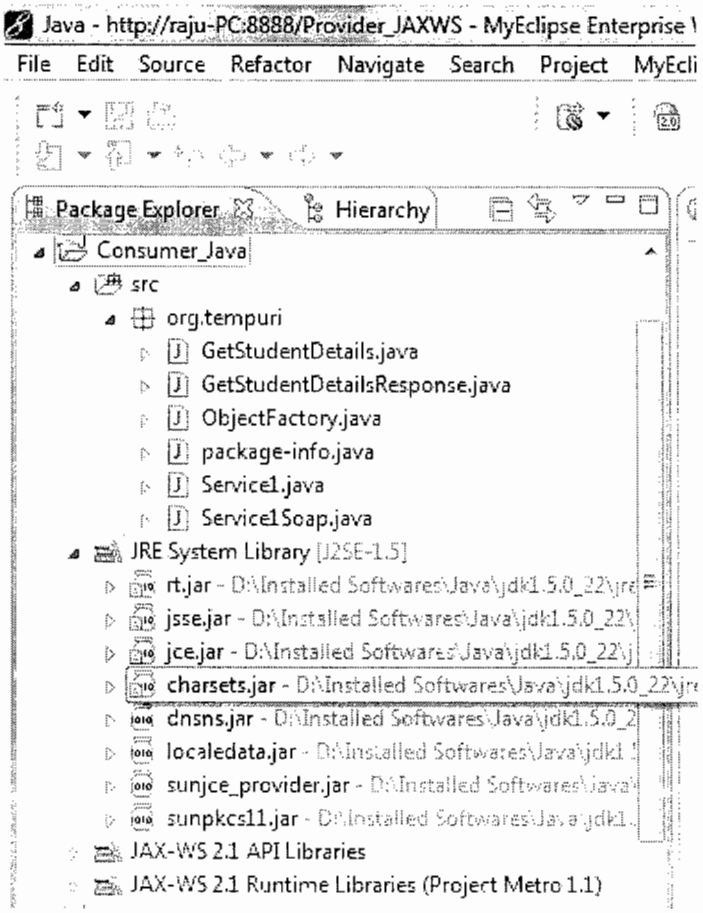
Step 4: Choose Web Service Client-->Click on Next.



Step 5: Enter the URL of WSDL (Provider_.Net)



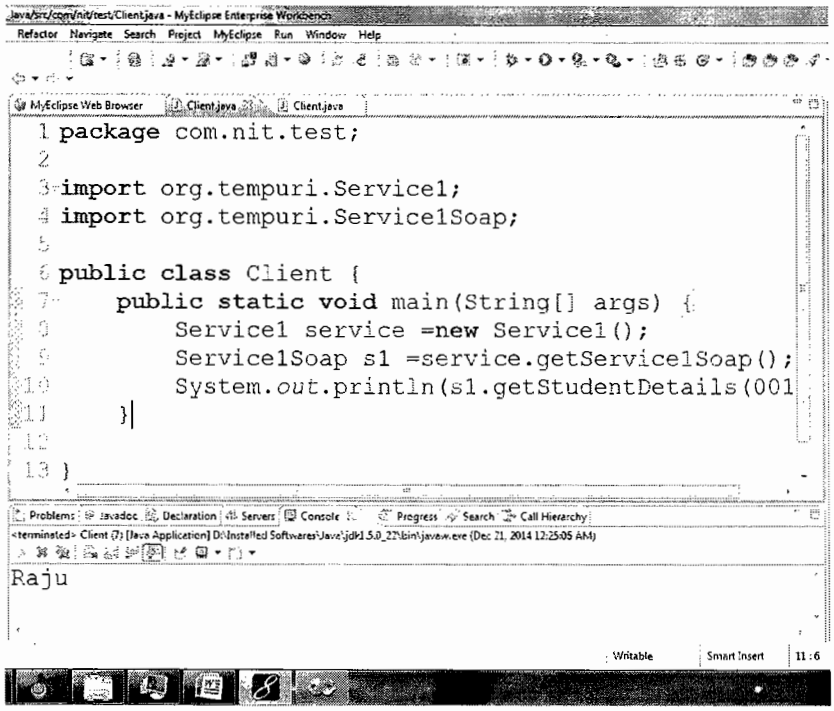
Step 5: Click on Next-->Next-->Finish.
Step 6: Find the generated Client-Side artifacts.



Step 7: Write a Client Program to test application using generated classes.

```
package com.nit.test;
import org.tempuri.Service1;
import org.tempuri.Service1Soap;
public class Client {
        public static void main(String[] args) {
                Service1 service =new Service1();
                Service1Soap s1 =service.getService1Soap();
                System.out.println(s1.getStudentDetails(001));
        }
}
```

Step 8: Run the Client and observe the Result.



JAX-WS RI(Reference Implementation)

PROVIDER:

1. Create a Dynamic Project in Eclipse Project.
2. Copy the jar files from JAXWS2.2.6-20120220\jaxws-ri\lib(Download it)
3. Code interface and implementation logic.
4. Create sun-jaxws.xml file
5. Modify web.xml with WSServlet.
6. Set the path for wsgen tool for jdk1.6 and run the following command.
7. wsgen –cp WebRoot/WEB-INF/classes -d src -keep -verbose com.nit.service.StudentImpl
8. It will generate the server side artifacts
9. Deploy into the sever and test.

a)Interface

```
package com.nit.service;
import javax.jws.WebMethod;
import javax.jws.WebService;
import javax.xml.ws.WebServiceException;
@WebService
public interface IStudent {
        @WebMethod
        public String getStudentDetails(Integer sno)throws WebServiceException;
}
```

b)Implementation Class

```
package com.nit.service;
import javax.jws.WebMethod;
import javax.jws.WebService;
import javax.xml.ws.WebServiceException;
@WebService
public class StudentImpl implements IStudent{
   @WebMethod
        public String getStudentDetails(Integer sno) throws WebServiceException {
                // TODO Auto-generated method stub
         String name =null;
         if(sno == 001){
                 name= "Raju";
         }else if(sno ==002){
                 name = "Rani";
         }else{
                 name ="Mantri";
         }
                return name;
        }
}
```

c)sun-jaxws.xml

```
<?xml version = "1.0"?>
<endpoints version="2.0"
        xmlns="http://java.sun.com/xml/ns/jax-ws/ri/runtime">
        <endpoint name="StudentImplPort"
                implementation="com.nit.service.StudentImpl"
                url-pattern="/StudentImplPort">
        </endpoint>
</endpoints>
```

d)web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.5"
        xmlns="http://java.sun.com/xml/ns/javaee"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
        http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd">
 <servlet>
        <description>JAX-WS endpoint - StudentImplService</description>
        <display-name>StudentImplService</display-name>
        <servlet-name>StudentImplService</servlet-name>
        <servlet-class>
                com.sun.xml.ws.transport.http.servlet.WSServlet
        </servlet-class>
        <load-on-startup>1</load-on-startup>
 </servlet>
 <servlet-mapping>
        <servlet-name>StudentImplService</servlet-name>
        <url-pattern>/StudentImplPort</url-pattern>
 </servlet-mapping>
 <listener>
        <listener-class>
                com.sun.xml.ws.transport.http.servlet.WSServletContextListener
        </listener-class>
 </listener>
</web-app>
```

CONSUMER:

1. create a java project
2. go to the location in command prompt and run the command
3. wsimport –d src –keep –verbose http://localhost:8080/jaxwsstudent/getDetails?wsdl

JAX-WS with Security

PROVIDER:

IStudent.java

```
package com.nit.service;
import javax.jws.WebMethod;
import javax.jws.WebService;
import javax.jws.soap.SOAPBinding;
import javax.jws.soap.SOAPBinding.Style;
@WebService
@SOAPBinding(style = Style.RPC)
public interface IStudent {
        @WebMethod
        String getStudentDetails(Integer sno);
}
```

StudentImpl.java

```
package com.nit.service;
import java.util.List;
import java.util.Map;
import javax.annotation.Resource;
import javax.jws.WebService;
import javax.xml.ws.WebServiceContext;
import javax.xml.ws.handler.MessageContext;
@WebService(endpointInterface = "com.nit.service.IStudent")
public class StudentImpl implements IStudent {
        public String getStudentDetails(Integer sno) {
                String sname = null;
                boolean isValid = isValidUser();
                if (isValid) {
                        if (sno == 001) {
                                sname = "Raju";
                        } else {
                                sname = "Mantri";
                        }
                        return sname;
                } else {
                        return "Not Valid User";
                }

        }

        @Resource
        WebServiceContext webServiceContext;
```

```
        public boolean isValidUser() {
                MessageContext messageContext = webServiceContext.getMessageContext();
                // get request headers
                Map<?, ?> requestHeaders = (Map<?, ?>) messageContext
                                .get(MessageContext.HTTP_REQUEST_HEADERS);
                List<?> usernameList = (List<?>) requestHeaders.get("username");
                List<?> passwordList = (List<?>) requestHeaders.get("password");
                String username = "";
                String password = "";
                if (usernameList != null) {
                        username = usernameList.get(0).toString();
                }
                if (passwordList != null) {
                        password = passwordList.get(0).toString();
                }
                if (username.equals("raju") && password.equals("secret")) {
                        return true;
                } else {
                        return false;
                }
        }
}
```

WebServicePublisher.java

```
package com.nit.service;
import javax.xml.ws.Endpoint;
import com.nit.service.StudentImpl;
public class WebServicePublisher{
        public static void main(String[] args) {
          Endpoint.publish("http://localhost:8888/WSSecurity/StudentService", new StudentImpl());
          System.out.println("Service Published:");
   }
}
```

WebServiceClient.java

```
package com.nit.service.client:
import java.net.URL;
import java.util.Collections;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

import javax.xml.namespace.QName;
import javax.xml.ws.BindingProvider;
import javax.xml.ws.Service;
```

```
import javax.xml.ws.handler.MessageContext;
import com.nit.service.IStudent;
public class WebServiceClient{
        public static void main(String[] args) throws Exception {
          URL wsdlUrl = new URL("http://localhost:8888/WSSecurity/StudentService?wsdl");
          //qualifier name ...
    QName qname = new QName("http://service.nit.com/", "StudentImplService");
    Service service = Service.create(wsdlUrl, qname);
    IStudent istudent = service.getPort(IStudent.class);
    Map<String, Object> requestContext = ((BindingProvider)istudent).getRequestContext();
            requestContext.put(BindingProvider.ENDPOINT_ADDRESS_PROPERTY,
                "http://localhost:8888/WSSecurity/StudentService?wsdl");
    Map<String, List<String>> requestHeaders = new HashMap<String, List<String>>();
    requestHeaders.put("username", Collections.singletonList("raju"));
    requestHeaders.put("Password", Collections.singletonList("secret"));
    requestContext.put(MessageContext.HTTP_REQUEST_HEADERS, requestHeaders);
    System.out.println(istudent.getStudentDetails(001));
  }
}
```

JAX-WS -Handlers

Handlers are interceptors that can be easily plugged into the Java API for XML-Based Web Services (JAX-WS) 2.0 runtime environment to do additional processing of inbound and outbound messages.

Web Services and their clients may need to access the SOAP message for additional processing of the message request or response. You can create SOAP message handlers to enable Web Services and clients to perform this additional processing on the SOAP message. A SOAP message handler provides a mechanism for intercepting the SOAP message in both the request and response of the Web Service.

A simple example of using handlers is to access information in the header part of the SOAP message. You can use the SOAP header to store Web Service specific information and then use handlers to manipulate it.

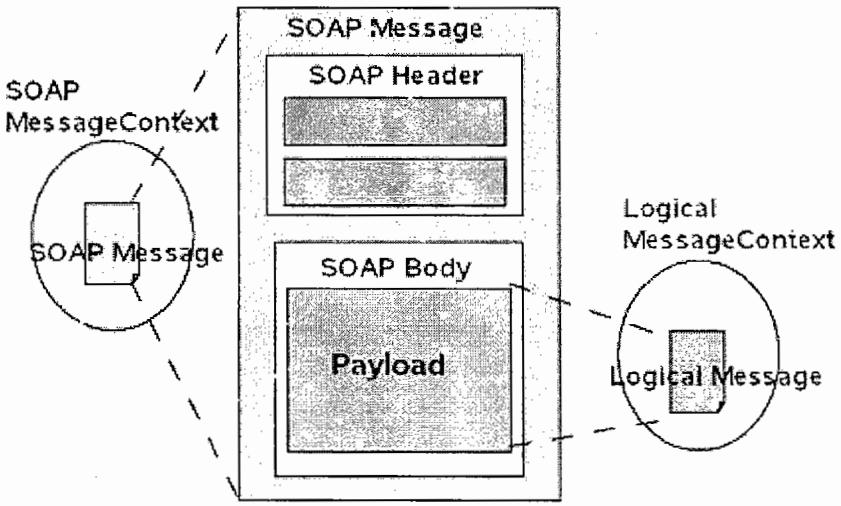
You can also use SOAP message handlers to improve the performance of your Web Service. After your Web Service has been deployed for a while, you might discover that many consumers invoke it with the same parameters.

JAX-WS supports two types of SOAP message handlers:

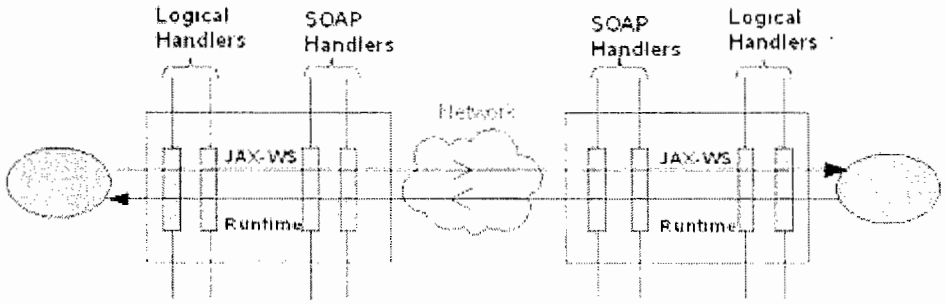
1. SOAP handlers and
2. Logical handlers.

SOAP handlers can access the entire SOAP message, including the message headers and body.
Logical handlers can access the payload of the message only, and cannot change any protocol-specific information (like headers) in a message

SOAP Format:



Handler Chain:



Types of SOAP Message Handlers:

1.SOAP handler	Enables you to access the full SOAP message including headers. SOAP handlers are defined using the javax.xml.ws.handler.soap.SOAPHandler interface. They are invoked using the import javax.xml.ws.handler.soap.SOAPMessageContext which extends javax.xml.ws.handler.MessageContext. The SOAPMessageContext.getMessage() method returns ajavax.xml.soap.SOAPMessage.
2.Logical handlers	Provides access to the payload of the message. Logical handlers cannot change any protocol-specific information (like headers) in a message. Logical handlers are defined using thejavax.xml.ws.handler.LogicalHandler interface. They are invoked using the javax.xml.ws.handler.LogicalMessageContext which extendsjavax.xml.ws.handler.MessageContext. The LogicalMessageContext.getMessage() method returns a javax.xml.ws.LogicalMessage. The payload can be accessed either as a JAXB object or as a javax.xml.transform.Source object.

Each type of message handler extends the javax.xml.ws.Handler interface which defines the methods defined in the following table.

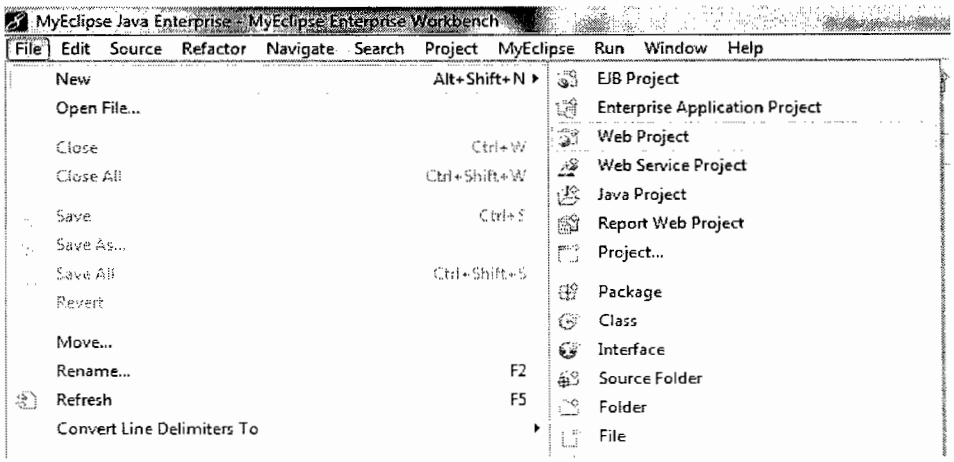
Handler Interface Methods:

Method:	Description
handleMessage()	Manages normal processing of inbound and outbound messages. A property in the MessageContext object is used to determine if the message is inbound or outbound. SeeImplementing the Handler.handleMessage() Method.
handleFault()	Manages fault processing of inbound and outbound messages. See Implementing the Handler.handleFault() Method.
close()	Concludes the message exchange and cleans up resources that were accessed during processing. See Implementing the Handler.close() Method.

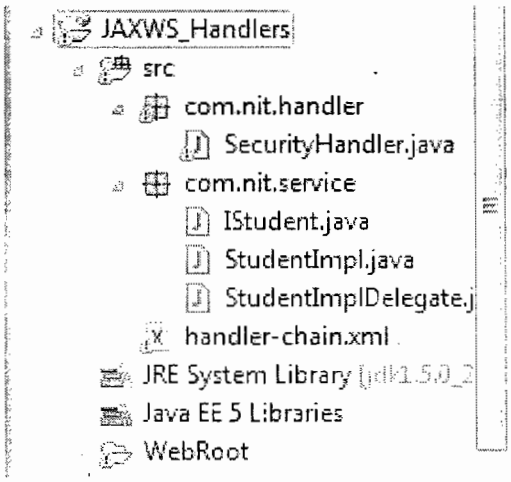
Example of SOAP Handler:

PROVIDER:

Step 1: Create Web Project (JAXWS_Handlers)



Step 2: Write an SEI interface and Implementation classes.



Step 3:write an SEI interface

IStudent.java

```
package com.nit.service;
import javax.jws.WebMethod;
import javax.jws.WebService;
import javax.xml.ws.WebServiceException;
@WebService
public interface IStudent {
        @WebMethod
        public String getStudentDetails(Integer sno)throws WebServiceException;
}
```

Step 4:write an implementation class

a)include @HandlerChain(file ="handler-chain.xml")

StudentImpl.java

```
package com.nit.service;


import javax.jws.HandlerChain;
import javax.jws.WebMethod;
import javax.jws.WebService;
```

```
import javax.xml.ws.WebServiceException;

@WebService(endpointInterface = "com.nit.service.IStudent")
@HandlerChain(file = "handler-chain.xml")
public class StudentImpl implements IStudent {
	@WebMethod
	public String getStudentDetails(Integer sno) throws WebServiceException {
		// TODO Auto-generated method stub

		String name = null;
		if (sno == 001) {
			name = "Raju";
		} else if (sno == 002) {
			name = "Rani";
		} else {
			name = "Mantri";
		}
		return name;
	}

}
```

Step 5: write an handler class.

```
package com.nit.handler;
import java.io.IOException;
import java.util.Iterator;
import java.util.Set;
import javax.xml.namespace.QName;
import javax.xml.soap.Node;
import javax.xml.soap.SOAPBody;
import javax.xml.soap.SOAPConstants;
import javax.xml.soap.SOAPEnvelope;
import javax.xml.soap.SOAPException;
import javax.xml.soap.SOAPFault;
import javax.xml.soap.SOAPHeader;
import javax.xml.soap.SOAPMessage;
import javax.xml.ws.handler.MessageContext;
import javax.xml.ws.handler.soap.SOAPHandler;
import javax.xml.ws.handler.soap.SOAPMessageContext;
import javax.xml.ws.soap.SOAPFaultException;
public class SecurityHandler implements SOAPHandler<SOAPMessageContext> {
	public boolean handleMessage(SOAPMessageContext context) {
		System.out.println("Server : handleMessage()......");
```

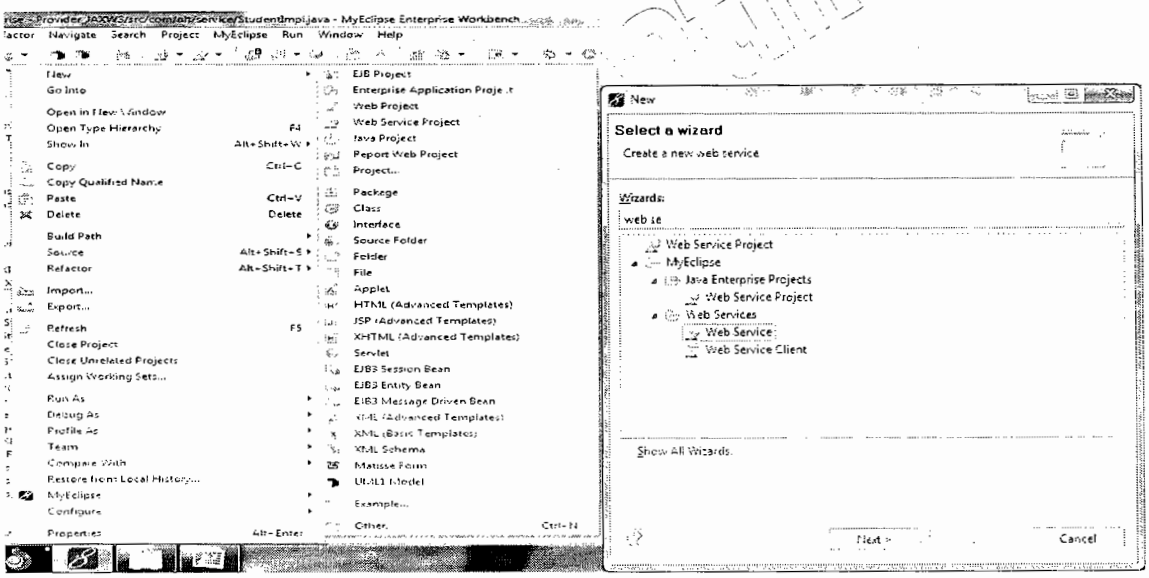
```
                try {
Boolean outBoundProperty = (Boolean) context
        .get(MessageContext.MESSAGE_OUTBOUND_PROPERTY);
        if (!outBoundProperty) {
                SOAPMessage soapMsg = context.getMessage();
                SOAPEnvelope soapEnv = soapMsg.getSOAPPart().getEnvelope();
                SOAPHeader soapHeader = soapEnv.getHeader();
                // if no header, add one
                  if (soapHeader == null) {
                        soapHeader = soapEnv.addHeader();
                        generateSOAPErrMessage(soapMsg, "No SOAP header.");
                }

                // Get client password from SOAP header
         Iterator it = soapHeader .extractHeaderElements(SOAPConstants.URI_SOAP_ACTOR_NEXT);
        Node password = (Node) it.next();
        String passwordValue = (password == null) ? null : password.getValue();
                if (passwordValue == null) {
                   generateSOAPErrMessage(soapMsg,"No password in header block.");
                }
                // if password is not match, throw exception
                if (!passwordValue.equals("Secret")) {
                 generateSOAPErrMessage(soapMsg,"Invalid password , Access is denied.");
                }
                soapMsg.writeTo(System.out);
            }
                } catch (SOAPException e) {
                        System.err.println(e);
                } catch (IOException e) {
                        System.err.println(e);
                }
                return true;
        }
        public boolean handleFault(SOAPMessageContext context) {
                System.out.println("Server : handleFault()......");
                return true;
        }
        public void close(MessageContext context) {
                System.out.println("Server : close()......");
        }
        public Set<QName> getHeaders() {
                System.out.println("Server : getHeaders()......");
                return null;
        }
```

```
        private void generateSOAPErrMessage(SOAPMessage msg, String reason) {
                try {
                        SOAPBody soapBody = msg.getSOAPPart().getEnvelope().getBody();
                        SOAPFault soapFault = soapBody.addFault();
                        soapFault.setFaultString(reason);
                        throw new SOAPFaultException(soapFault);
                } catch (SOAPException e) {
                }
        }
}
```

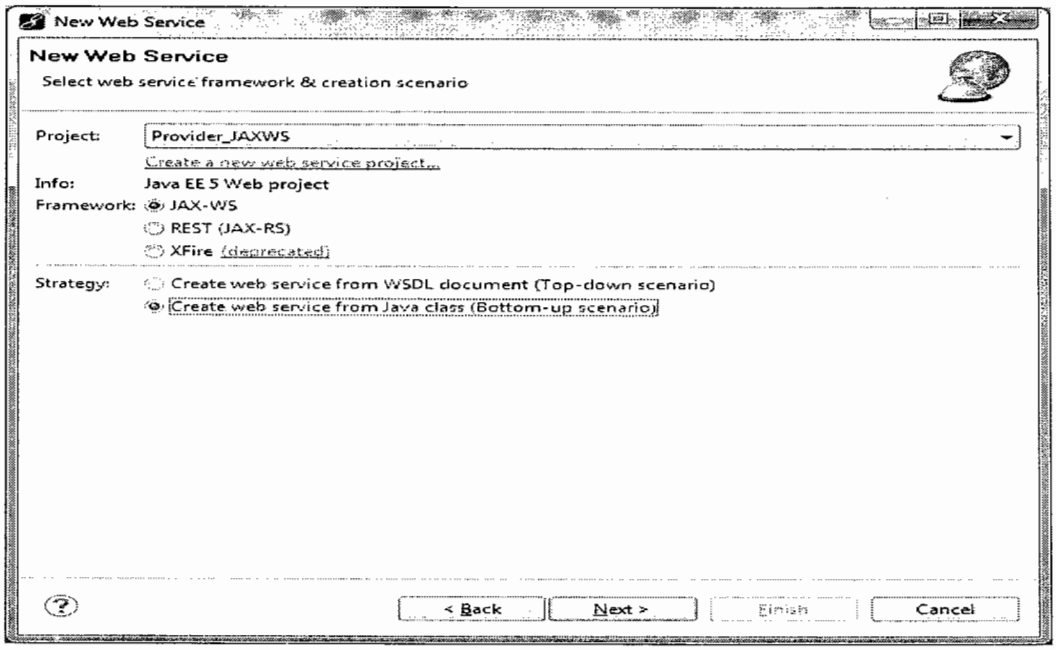
Step 6:handler-chain.xml

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<handler-chains
    xmlns="http://java.sun.com/xml/ns/javaee"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <handler-chain>
    <handler>
      <handler-class>com.nit.handler.SecurityHandler</handler-class>
    </handler>
  </handler-chain>
</handler-chains>
```

Step 7: Right Click on project , choose New-->Other-->Web Service

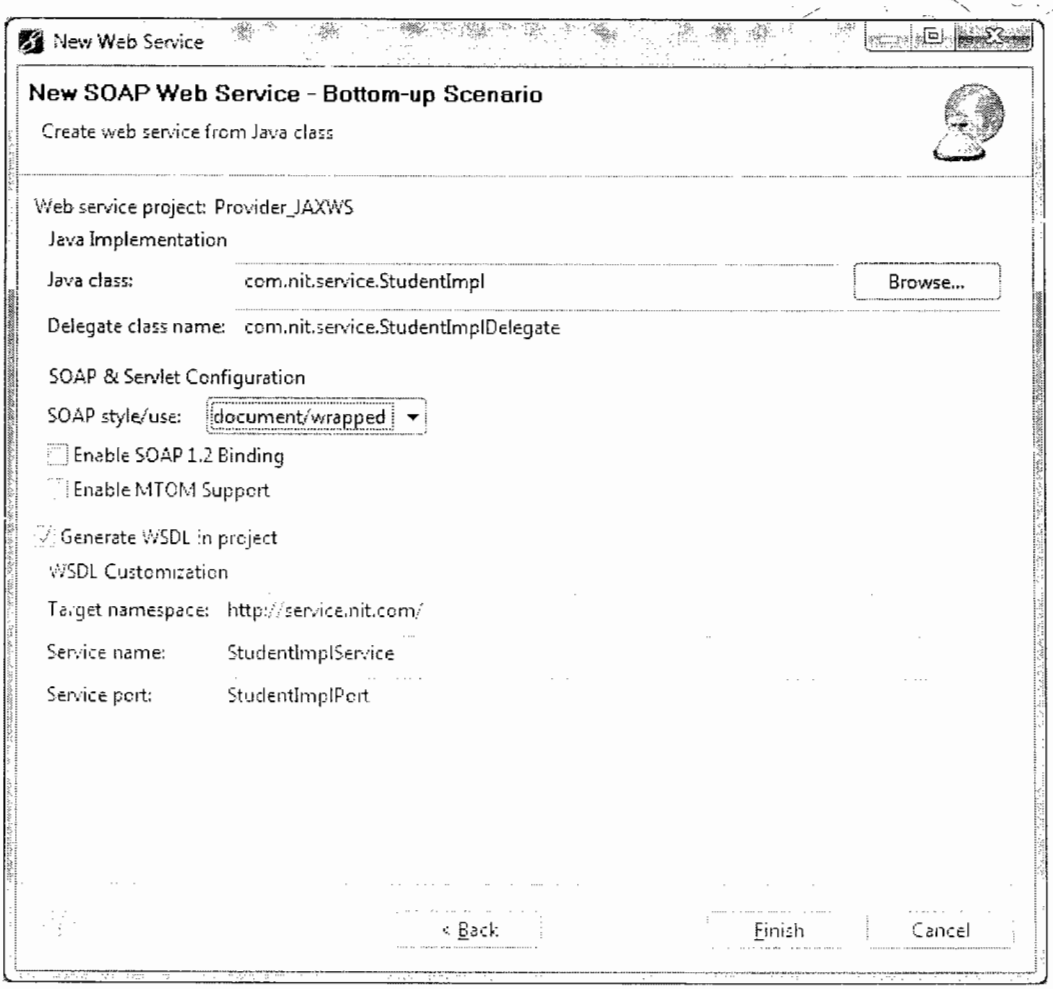


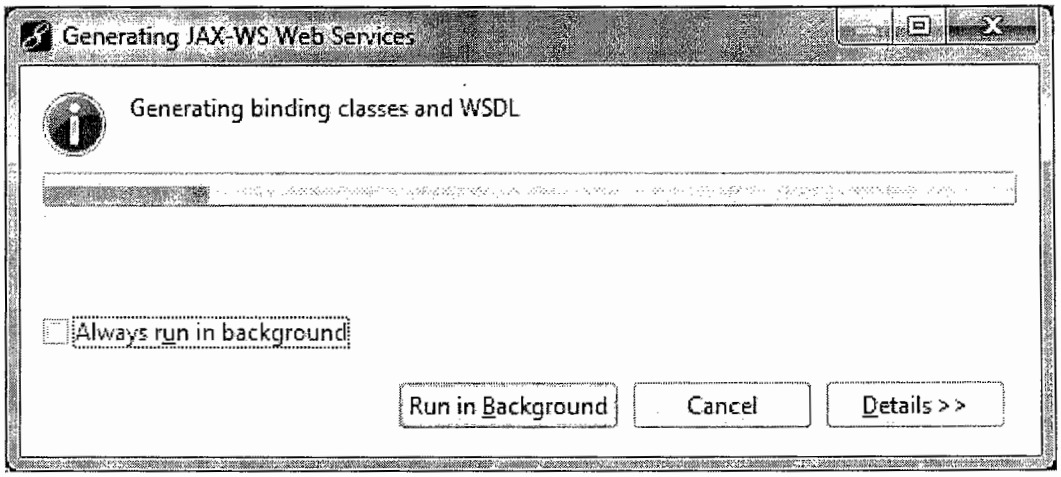
Step7: Choose JAX-WS and Bottom-Up Scenario as shown below.



Step 8: Follow the below steps:

1. Choose our Implementation class
2. Select message-exchange format (ex. document/wrapped)
3. Click on Check box-Generate WSDL in project
4. Observe Target Name space, Service Name, Service Port
   i. (If you want to change the names change it)
5. Click on Finish, It will generate Provider Artifacts as shown below.





Step 9: Observe the project structure.

1. It is modifying the web.xml as shown below.
2. Generating WSDL in the project.
3. Generated sun-jaxws.xml

web.xml

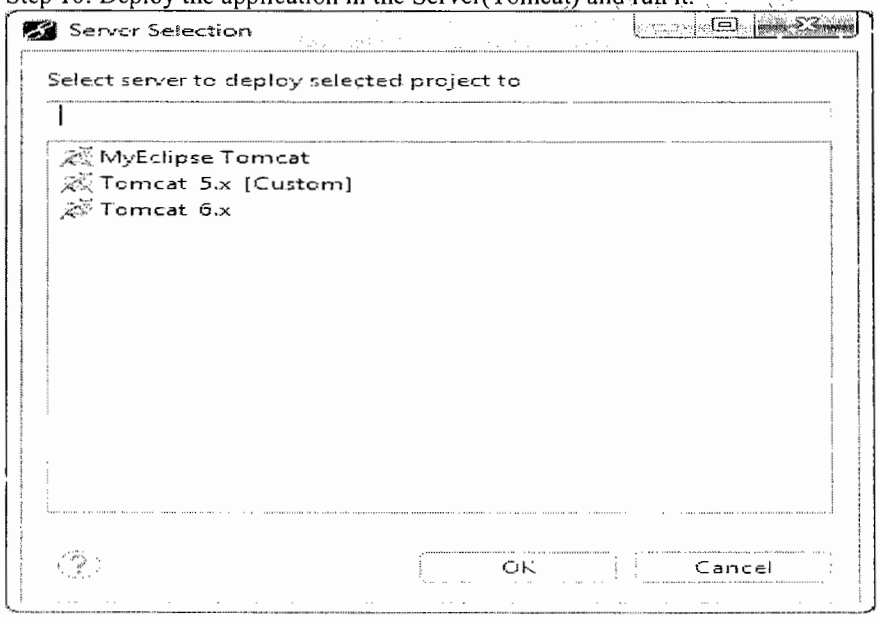
```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.5"
        xmlns="http://java.sun.com/xml/ns/javaee"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
        http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd">
  <servlet>
        <description>JAX-WS endpoint - StudentImplService</description>
        <display-name>StudentImplService</display-name>
        <servlet-name>StudentImplService</servlet-name>
        <servlet-class>
                com.sun.xml.ws.transport.http.servlet.WSServlet
        </servlet-class>
        <load-on-startup>1</load-on-startup>
  </servlet>
```

```
<servlet-mapping>
        <servlet-name>StudentImplService</servlet-name>
        <url-pattern>/StudentImplPort</url-pattern>
</servlet-mapping>
<welcome-file-list>
  <welcome-file>index.jsp</welcome-file>
</welcome-file-list>
<listener>
        <listener-class>
                com.sun.xml.ws.transport.http.servlet.WSServletContextListener
        </listener-class>
</listener></web-app>
```

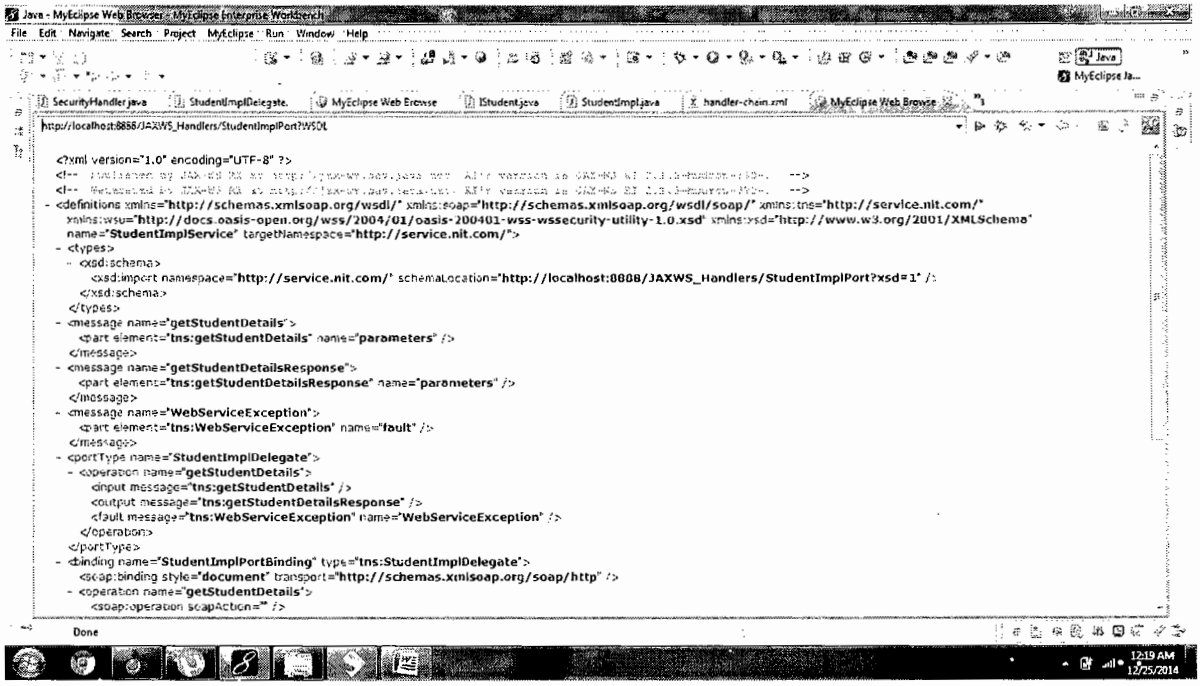
sun-jaxws.xml

```
<?xml version = "1.0"?>
<endpoints version="2.0"
        xmlns="http://java.sun.com/xml/ns/jax-ws/ri/runtime">
        <endpoint name="StudentImplPort"
                implementation="com.nit.service.StudentImplDelegate"
                url-pattern="/StudentImplPort">
        </endpoint></endpoints>
```

Step 10: Deploy the application in the Server(Tomcat) and run it.

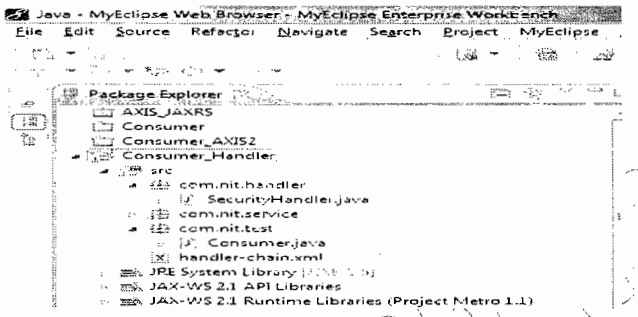


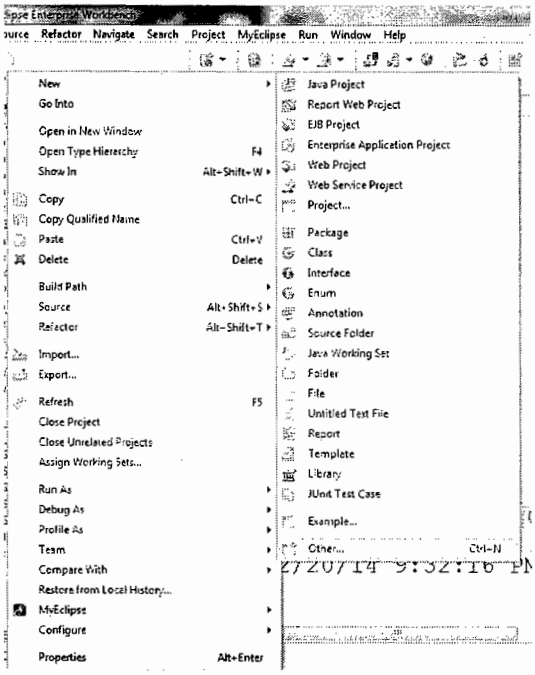
Step 11: Run the following URL and check the WSDL file.
http://localhost:8888/JAXWS_Handlers/StudentImplPort?WSDL



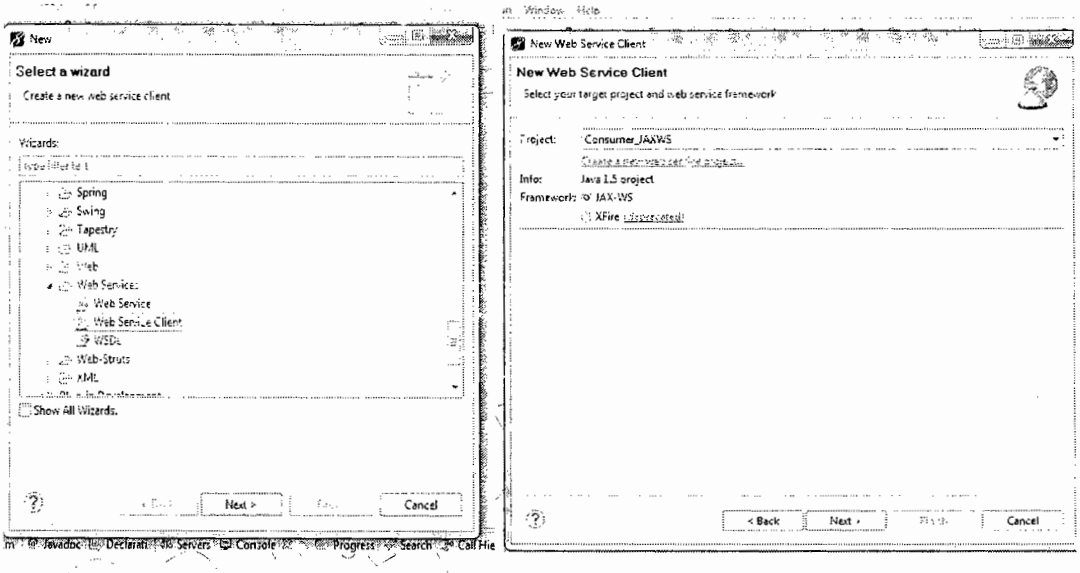
CONSUMER:

Step 1: Create a java project Consumer_Handlers

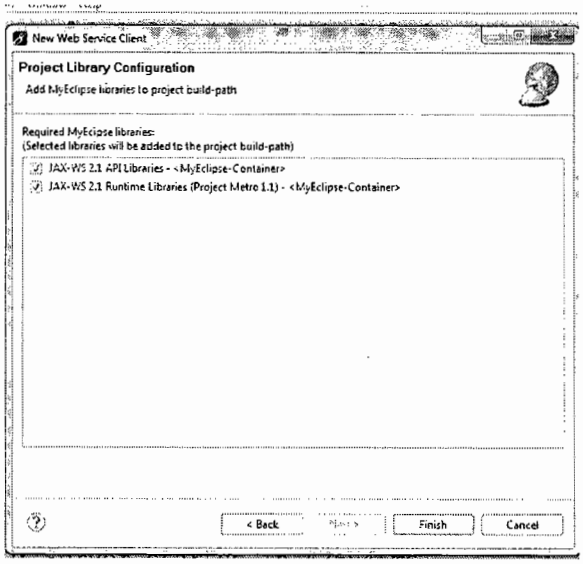




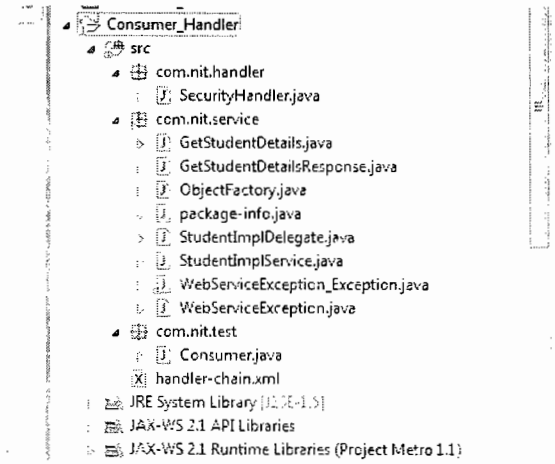
Step 2: Right click on project New-->Other-->Web Service Client



Step 3: Enter the URL of WSDL (which is running above Provider)
http://localhost:8888/Provider_Handlers/StudentImplPort?WSDL



Step 4: Observe the client-side artifacts as shown in below.



Step 5: modify the StudentImplService.java from generated client-side artifacts with @HandlerChain as shown below.

```
package com.nit.service;
import java.net.MalformedURLException;
import java.net.URL;
import java.util.logging.Logger;
import javax.jws.HandlerChain;
import javax.xml.namespace.QName;
import javax.xml.ws.Service;
import javax.xml.ws.WebEndpoint;
import javax.xml.ws.WebServiceClient;
@WebServiceClient(name = "StudentImplService", targetNamespace = "http://service.nit.com/",
wsdlLocation = "http://raju-pc:8888/JAXWS_Handlers/StudentImplPort?WSDL")
@HandlerChain(file = "handler-chain.xml")
public class StudentImplService extends Service {

        private final static URL STUDENTIMPLSERVICE_WSDL_LOCATION;
        private final static Logger logger = Logger
```

```
                    .getLogger(com.nit.service.StudentImplService.class.getName());

        static {
                URL url = null;
                try {
                URL baseUrl;
                baseUrl = com.nit.service.StudentImplService.class.getResource(".");
                url = new URL(baseUrl,
                "http://raju-pc:8888/JAXWS_Handlers/StudentImplPort?WSDL");
                } catch (MalformedURLException e) {
                logger.warning("Failed to create URL for the wsdl Location: 'http://raju-
pc:8888/JAXWS_Handlers/StudentImplPort?WSDL', retrying as a local file");
                logger.warning(e.getMessage());
                }
                STUDENTIMPLSERVICE_WSDL_LOCATION = url;
        }
        public StudentImplService(URL wsdlLocation, QName serviceName) {
                super(wsdlLocation, serviceName);
        }
        public StudentImplService() {
                super(STUDENTIMPLSERVICE_WSDL_LOCATION, new QName(
                                "http://service.nit.com/", "StudentImplService"));
        }
        @WebEndpoint(name = "StudentImplPort")
        public StudentImplDelegate getStudentImplPort() {
                return super.getPort(new QName("http://service.nit.com/",
                                "StudentImplPort"), StudentImplDelegate.class);
        }

}
```

Step 6: write an handler java file
SecurityHandler.java

```
package com.nit.handler;
import java.io.IOException;
import java.util.Set;
import javax.xml.namespace.QName;
import javax.xml.soap.SOAPConstants;
import javax.xml.soap.SOAPEnvelope;
import javax.xml.soap.SOAPException;
import javax.xml.soap.SOAPHeader;
import javax.xml.soap.SOAPHeaderElement;
import javax.xml.soap.SOAPMessage;
import javax.xml.ws.handler.MessageContext;
import javax.xml.ws.handler.soap.SOAPHandler;
import javax.xml.ws.handler.soap.SOAPMessageContext;
```

```
public class SecurityHandler implements SOAPHandler<SOAPMessageContext> {
    public boolean handleMessage(SOAPMessageContext context) {
        System.out.println("Client : handleMessage()......");
        try {
            Boolean outboundProperty = (Boolean) context
                .get(MessageContext.MESSAGE_OUTBOUND_PROPERTY);
                if (outboundProperty.booleanValue()) {
                SOAPMessage soapMsg = context.getMessage();
                SOAPEnvelope soapEnv = soapMsg.getSOAPPart().getEnvelope();
                        SOAPHeader soapHeader = soapEnv.getHeader();
                        // if no header, add one
                        if (soapHeader == null) {
                                soapHeader = soapEnv.addHeader();
                        }
                        // get padd address
                        String password = "Secret";
                        // add a soap header, name as "password address"
                QName qname = new QName("http://service.nit.com/", "password");
                SOAPHeaderElement soapHeaderElement = soapHeader
                                        .addHeaderElement(qname);

soapHeaderElement.setActor(SOAPConstants.URI_SOAP_ACTOR_NEXT);
                        soapHeaderElement.addTextNode(password);
                        soapMsg.saveChanges();
                        // tracking
                        soapMsg.writeTo(System.out);
                }
        } catch (SOAPException e) {
                System.err.println(e);
                return false;
        } catch (IOException e) {
                System.err.println(e);
                return false;
        }
        return true;
    }
    public boolean handleFault(SOAPMessageContext context) {
        System.out.println("Client : handleFault()......");
        return true;
    }
    public void close(MessageContext context) {
        System.out.println("Client : close()......");
    }
```

```
	public Set<QName> getHeaders() {
		System.out.println("Client : getHeaders()......");
		return null;
	}

}
```

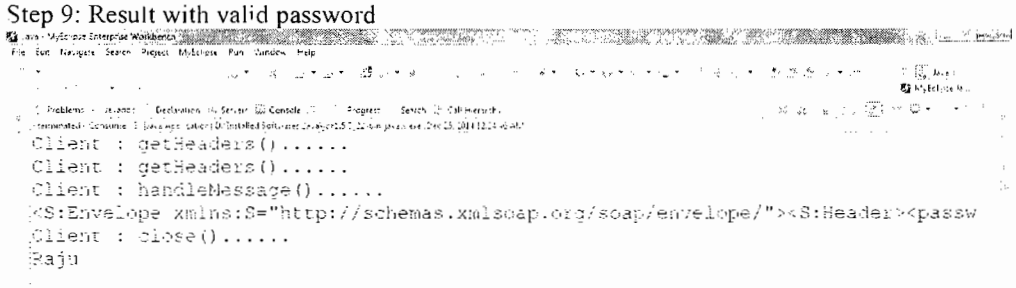
Step 7: HandlerChain.xml

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<handler-chains
   xmlns ="http://java.sun.com/xml/ns/javaee"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema">
 <handler-chain>
  <handler>
   <handler-class>com.nit.handler.SecurityHandler</handler-class>
  </handler>
 </handler-chain>
</handler-chains>
```

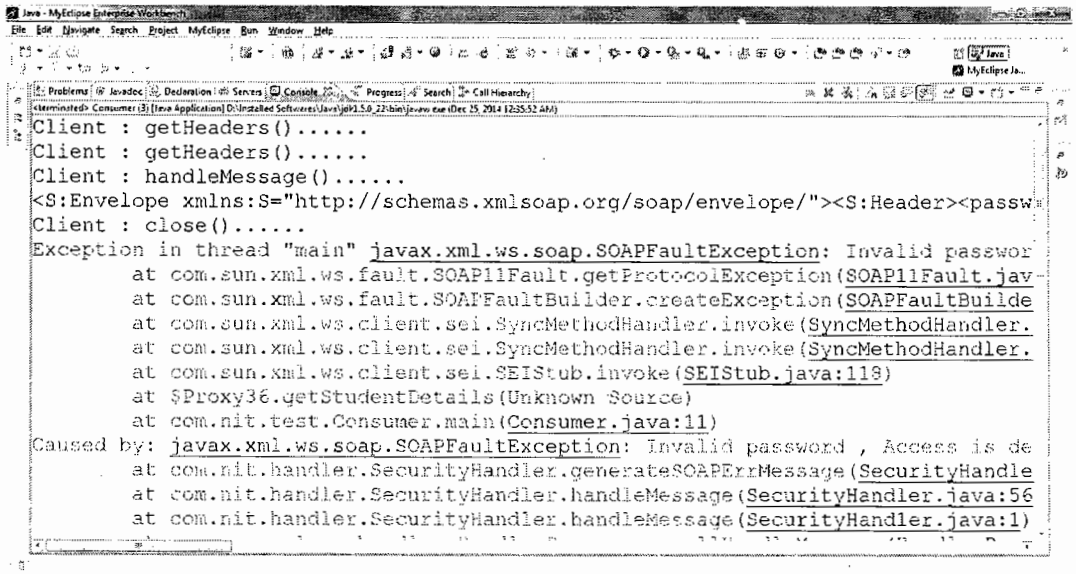
Step 8: Write a Client class to test the application using client-side artifacts.

```
package com.nit.test;

import com.nit.service.StudentImplDelegate;

import com.nit.service.StudentImplService;

import com.nit.service.WebServiceException_Exception;


public class Client {


	public static void main(String[] args) throws WebServiceException_Exception {
		StudentImplService service = new StudentImplService();
		StudentImplDelegate portType = service.getStudentImplPort();
		System.out.println(portType.getStudentDetails(001));
	}


}
```

Step 9: Result with valid password



Step 10: Result with in-valid password


Client : getHeaders()......
Client : getHeaders()......
Client : handleMessage()......
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/"><S:Header><passw
Client : close()......
Exception in thread "main" javax.xml.ws.soap.SOAPFaultException: Invalid passwor
at com.sun.xml.ws.fault.SOAP11Fault.getProtocolException(SOAP11Fault.jav
at com.sun.xml.ws.fault.SOAPFaultBuilder.createException(SOAPFaultBuilde
at com.sun.xml.ws.client.sei.SyncMethodHandler.invoke(SyncMethodHandler.
at com.sun.xml.ws.client.sei.SyncMethodHandler.invoke(SyncMethodHandler.
at com.sun.xml.ws.client.sei.SEIStub.invoke(SEIStub.java:118)
at $Proxy36.getStudentDetails(Unknown Source)
at com.nit.test.Consumer.main(Consumer.java:11)
Caused by: javax.xml.ws.soap.SOAPFaultException: Invalid password , Access is de
at com.nit.handler.SecurityHandler.generateSOAPErrMessage(SecurityHandle
at com.nit.handler.SecurityHandler.handleMessage(SecurityHandler.java:56
at com.nit.handler.SecurityHandler.handleMessage(SecurityHandler.java:1)

JAX-WS with Axis-2

PROVIDER:

1.Create a folder called **Provider**.
2.Create a java file called **StudentService.java**

```
public class StudentService
{
  public String getStudentDetails(Integer sno){
     String name =null;
     if(sno ==001){
       name ="Raju";
     }
     return name;
  }
}
```

3. In **"Provider"** folder create a new folder and name it as "META-INF".
4. Inside **META-INF** folder create **services.xml**
For Example:

```
<service>
<parameter name="ServiceClass" locked="false">StudentService</parameter>
<operation name="getStudentDetails">
        <messageReceiver class="org.apache.axis2.rpc.receivers.RPCMessageReceiver"/>
</operation>
</service>
```

5. Inside **Provider folder** create another new folder and name it as **"temp".**

6. Open the **command prompt** in Provider folder (Shift +Right Click mouse).

7. Compile **StudentService.java** and move the **StudentService.class** file to **"temp"** directory

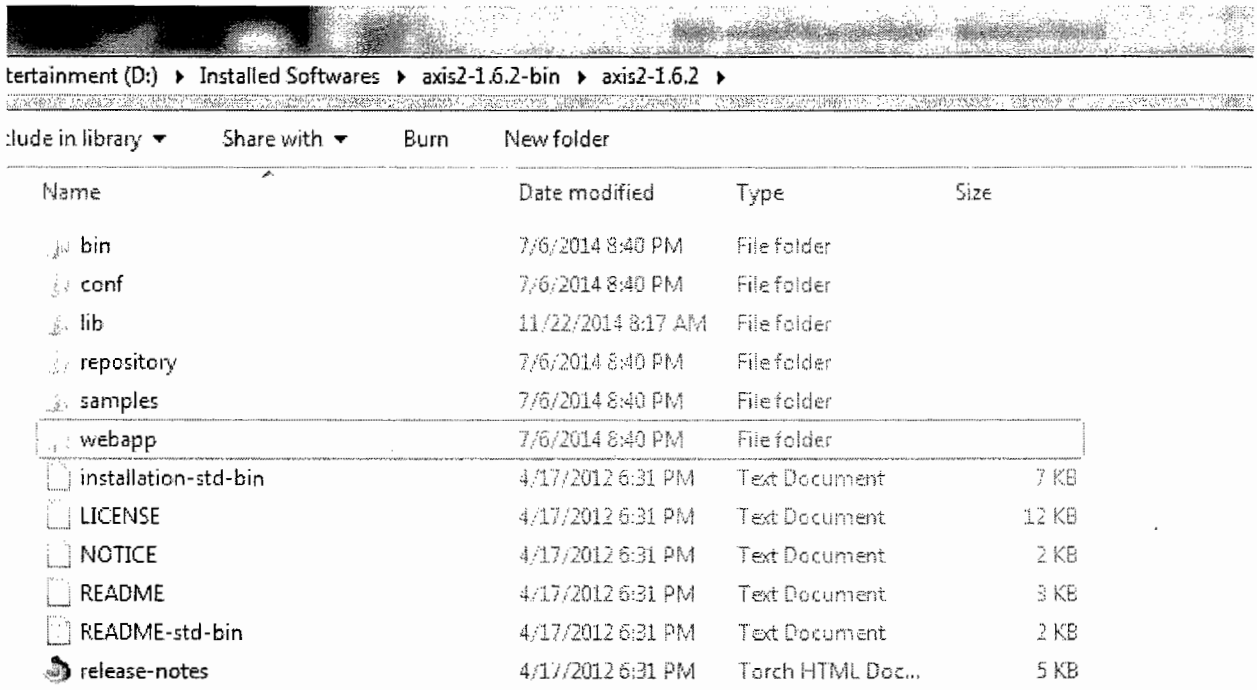
```
javac StudentService.java -d temp/
```

8. Copy **META-INF** folder which contains **services.xml** into **"temp"** directory.

9. Change directory to the "temp" directory and use the **"jar"** command as it follows to create archive named StudentService.aar

```
jar -cvf StudentService.aar *
```

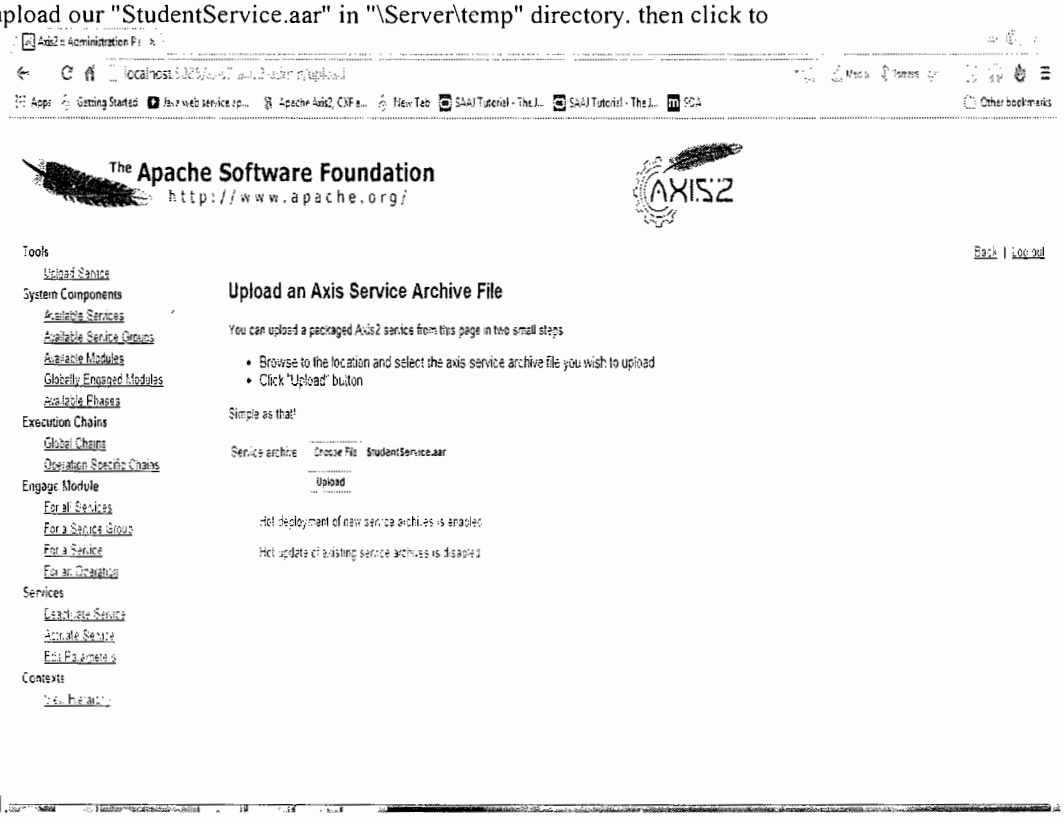
9. Copy axis2 folder into web-apps folder of tomcat(Download axis2 from net)



10.Using Axis2 with "Tomcat" to host start tomcat go to "**http://localhost:8080/axis2/axis2-admin**".

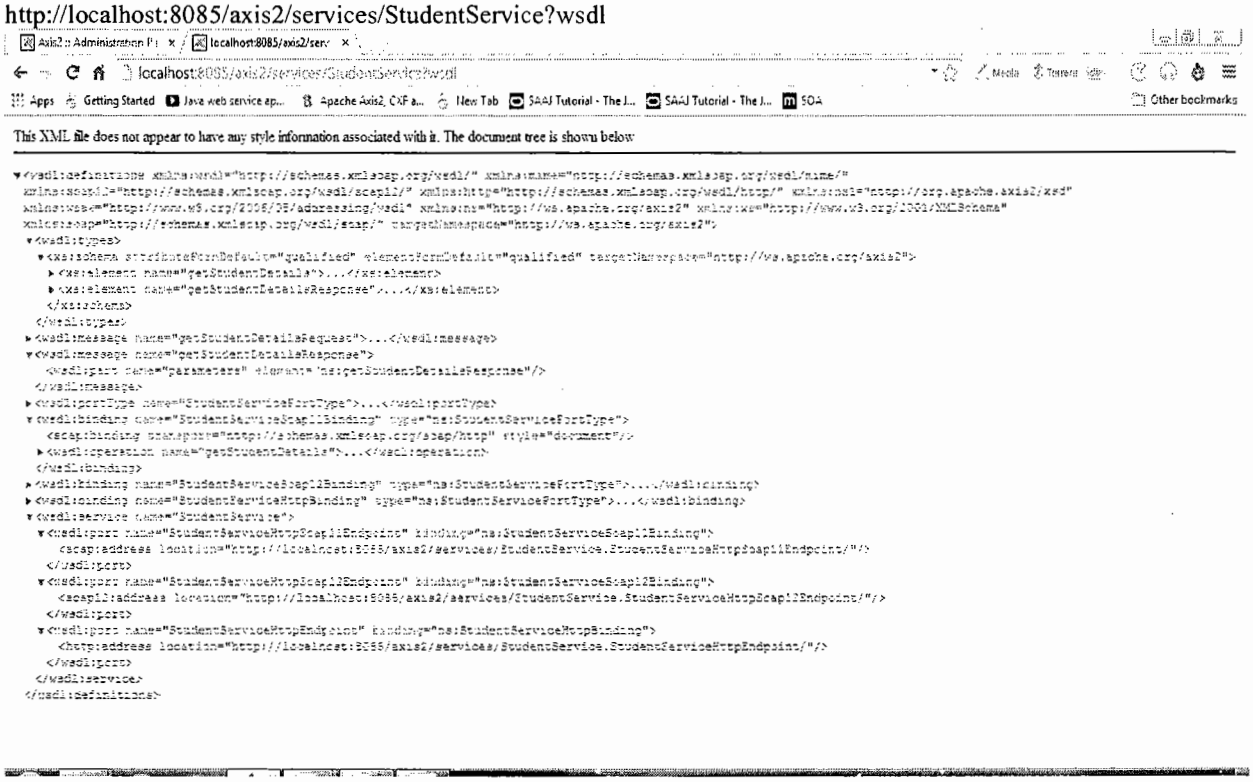
UserName : admin Password="axis2"

then upload our "StudentService.aar" in "\Server\temp" directory. then click to



upload.

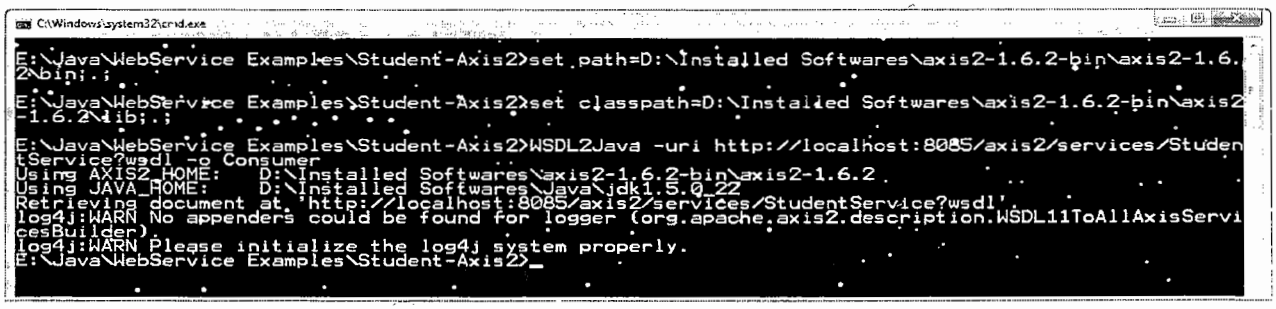
11. Go to the
http://localhost:8085/axis2/services/StudentService?wsdl



CONSUMER:

Step:1
E:\Java\WebService Examples\Student-Axis2>

WSDL2Java -uri http://localhost:8085/axis2/services/StudentService?wsdl -o Consumer



Step 2:

This generates two .java files and we will be using the **org.apache.axis2.StudentServiceStub** to invoke the "getStudentDetails" operation of the service inside the **Consumer** folder.

Step 3:

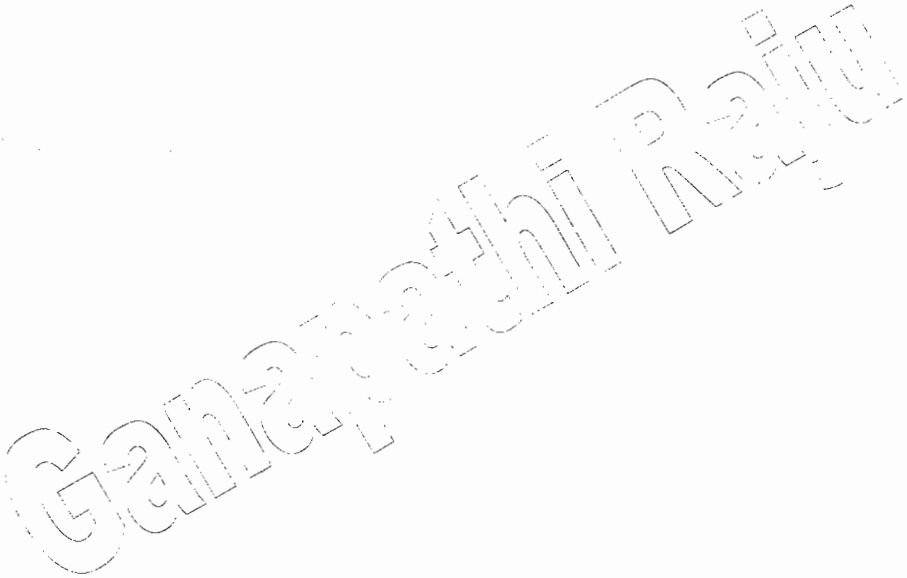
Now Create a new **Client.java** class which uses the org.apache.axis2.StudentServiceStub.

For simplicity lets create the Client.java file in the same package as the generated code (i.e.org.apache.axis) and save it along with the other generated code.

Client.java

```
package org.apache.axis2;

import org.apache.axis2.StudentServiceStub.GetDetailsStudent;

public class Client {
```

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

        StudentServiceStub stub = new StudentServiceStub();

        //Create the request
        StudentServiceStub. GetDetailsStudent request = new GetDetailsStudent.Echo();
        request.setParam0(001);

        //Invoke the service
        GetStudentDetailsResponse response = stub.getStudentDetails(request);

        System.out.println("Response : " + response.get_return());
    }

}
```



JAXWS-Apache CXF

Apache CXF is an open source services framework. CXF helps you build and develop services using frontend programming APIs, like JAX-WS and JAX-RS. These services can speak a variety of protocols such as SOAP, XML/HTTP, RESTful HTTP, or CORBA and work over a variety of transports such as HTTP.

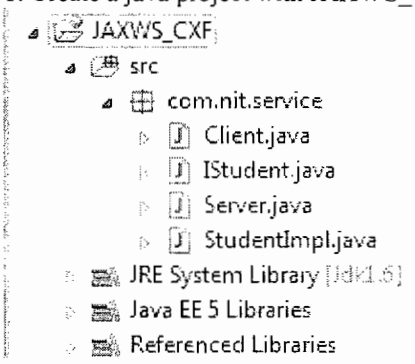
Features:

CXF includes a broad feature set, but it is primarily focused on the following areas:

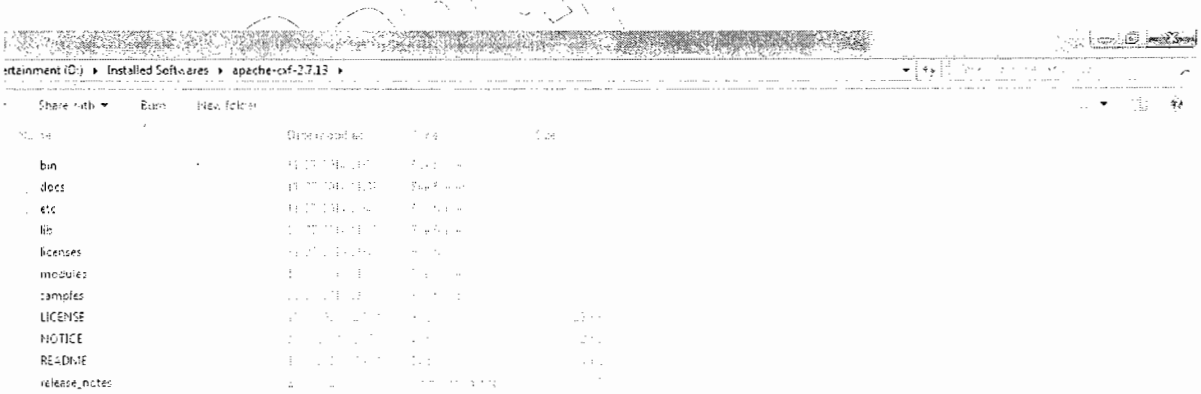
- **Web Services Standards Support:** CXF supports a variety of web service standards including SOAP, the WS-I Basic Profile, WSDL, WS-Addressing, WS-Policy, WS-ReliableMessaging, WS-Security, WS-SecurityPolicy, WS-SecureConverstation, and WS-Trust (partial).
- CXF implements the JAX-WS APIs. CXF JAX-WS support includes some extensions to the standard that make it significantly easier to use, compared to the reference implementation: It will automatically generate code for request and response bean classes, and does not require a WSDL for simple cases.
- It also includes a "simple frontend" which allows creation of clients and endpoints without annotations. CXF supports both contract first development with WSDL and code first development starting from Java.

PROVIDER:

Step 1: Create a java project with JAXWS_CXF



Step 2: Download Apache Axis-CXF from net.



Step 3: copy the jar files from apache-cxf lib folder into project
Step 4: Write the following files and execute it.

IStudent.java

```
package com.nit.service;
import javax.jws.WebParam;
import javax.jws.WebService;
@WebService
public interface IStudent {
  //@WebParam is optional
 String getMessage(@WebParam(name="text") String text);
}
```

StudentImpl.java

```
package com.nit.service;
import javax.jws.WebService;
@WebService
public class StudentImpl implements IStudent {
 public String getMessage(String name) {
   System.out.println("HI ! Welcome  " + name);
   return "Hello " + name;
 }
}
```

Server.java

```
package com.nit.service;
import org.apache.cxf.jaxws.JaxWsServerFactoryBean;
public class Server {
 private Server() {
   IStudent helloWorld = new StudentImpl();
   //create WebService service factory
   JaxWsServerFactoryBean factory = new JaxWsServerFactoryBean();
   //register WebService interface
   factory.setServiceClass(IStudent.class);
   //publish the interface
   factory.setAddress("http://localhost:8888/StudentService");
   factory.setServiceBean(helloWorld);
   //create WebService instance
   factory.create();
 }

 public static void main(String[] args) throws InterruptedException {
   //now start the webservice server
   new Server();
   System.out.println("Server ready...");
   Thread.sleep(1000 * 60);
   System.out.println("Server exit...");
```

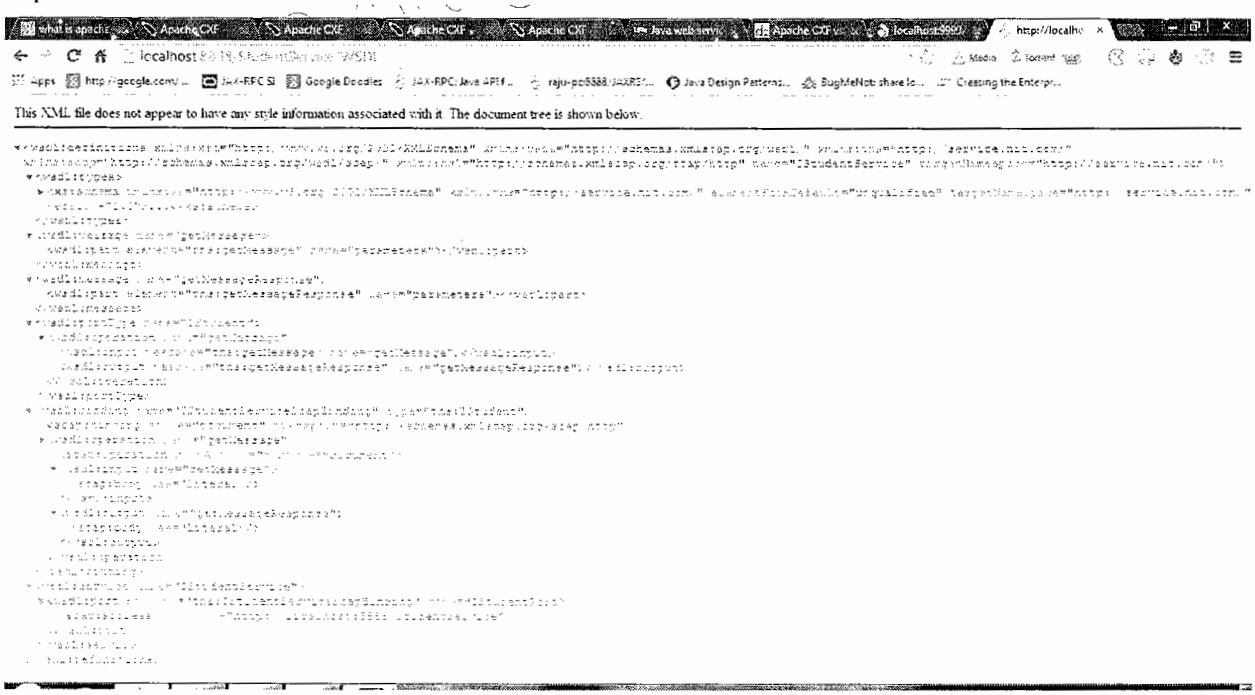
```
    System.exit(0);
  }
}
```

CONSUMER:

1.write a client

```
package com.nit.service;
import org.apache.cxf.jaxws.JaxWsProxyFactoryBean;

public class Client {
  private Client() {
  }

  public static void main(String[] args) {
    //create WebService client proxy factory
    JaxWsProxyFactoryBean factory = new JaxWsProxyFactoryBean();
    //register WebService interface
    factory.setServiceClass(IStudent.class);
    //set webservice publish address to factory.
    factory.setAddress("http://localhost:8888/StudentService");
    IStudent iStudent = (IStudent) factory.create();
    System.out.println("invoke webservice...");
    System.out.println("message context is:" + iStudent.getMessage("Raju"));
    System.exit(0);
  }
}
```

Hit the following URL for WSDL location
http://localhost:8888/StudentService?WSDL



Difference Between Apache Axis2, Apache CXF, Spring WS.

Framework	Key Positives	Key Concerns
Apache AXIS2	• Most Commonly Used, Matured & Stable Web Services Development Framework • Supports Multiple Languages (C++, Java) • Supports both Contract-first & Contract-last Approach • In context of Orchestration & Web Services Transaction (long-running transactions) it supports wide variety of related WS-* specifications: • WS-Atomic Transaction, WS-Business Activity, WS-Coordination, WS-Eventing, WS-Transfer Compatible with Spring Framework	• Comparatively More Code Required/Generated w.r.t. Spring WS/CXF • Is being phased out gradually (mostly by Apache CXF) • It is not fully compliant for JAX-WS JAX-RS
Apache-CXF	• Most widely used Web Services Standard Now; Improvement over AXIS2, which is now gradually being replaced by Apache CXF • Intuitive & Easy to Use (less coding required as compared to AXIS2) • Clean separation of front-ends, like JAX-WS, from the core code • Fully compliant with JAX-WS, JAX-RS & others • Best Performance across all available framework with minimum computation overhead • Supports wide variety of front-end models • Supports both JAX-WS & JAX-RS (for Restful Services) • Supports JBI & SDO (not supported in AXIS2) • Compatible with Spring Framework	• Does not support Orchestration & WS Transactions yet • Does not support WSDL 2.0 yet
Spring WS	• Best in terms of supporting Contract-first Web Services Development Approach • Enforces Standards & Best Practices by Framework Constraints (no way out of it & hence limitation as well) • Supports Spring Annotations as well as JAX-WS • Least code from developer's perspective • Best Aligned with Spring Technology Stack (also similar architectural stack as Spring MVC) including Spring Security	• Least number of WS-* Specifications supported (does not fully compliant with JAX-WS) • Spring offers itself as standard & hence other Java-compliant frameworks support better standards support • Only support Contract-first Web Services Development Model

Difference between JAX-RPC and JAX-WS

JAX-RPC	JAX-WS
WS-I's Basic Profile (BP) version 1.0	WS-I's Basic Profile (BP) version 1.1
JAX-RPC supports SOAP 1.1	JAX-WS supports SOAP 1.1 and SOAP 1.2
JAX-RPC supports WSDL 1.1 and Ignore Http Binding	JAX-WS supports WSDL 1.1 and 1.2(2.0) and also adds Http Binding
JAX-RPC maps to Java 1.4	JAX-WS maps to Java 5.0 on words
It doesn't support annotations	Annotation based webservice
JAX-RPC has its own data mapping model. i.e javax.xml.soap.SOAPElement.	JAX-WS's data mapping model is JAXB
JAX-RPC's is not supporting Asynchronous functionality.	JAX-WS's model introduces Asynchronous functionality.
JAX-RPC supports SAAJ(SOAP with Attachment API for Java)	MTOM (Message Transmission Optimization Mechanism)
JAX-RPC handlers rely on SAAJ 1.2.	JAX-WS handlers rely on the new SAAJ 1.3 specification.
Default encoding format is RPC-Encoded	No default need to choose.

Difference Between SOAP and Restful

SOAP(JAX-RPC & JAX-WS)	Restful
Heavy weight	Light weight
Security is more	Security is less
High bandwidth	Less bandwidth
Traditional	Recent
Difficult to learn/implement	Easier to learn/implement.
Cost of the product is high	Less cost
Sending and Receiving messages are complex	Sending and Receiving messages are easy
XML is the only primary business	It supports different formats of sending messages. Like HTML,XML,JSON,JPEG.

JAX-RS

- JAX-RS stands for Java API for XML RESTful Web services.
- REST stands for representational state transfer.
- JAX-RS is an API for implementing RESTful webservice.
- REST is an architectural style which is based on web-standards and the HTTP protocol.
- In REST everything is a resource.
- A resource is accessed via a common interface based on HTTP standard methods.
- REST allows that resources have different representations,
  ex. Text,XML,JSON,JPEG,
- Implementations of JAX-RS
  1. Jersey
  2. RESTEasy
  3. Restlet
  4. Apache CXF

What is REST?

- REST stands for Representational State Transfer.It relies on a stateless, client-server, cacheable communications protocol -- and in virtually all cases, the HTTP protocol is used.
- REST is an architecture style for designing networked applications. The idea is that, rather than using complex mechanisms such as CORBA, RPC or SOAP to connect between machines, simple HTTP is used to make calls between machines
- Platform-independent (you don't care if the server is Unix, the client is a Mac, or anything else),
- Language-independent (C# can talk to Java, etc.),
- Standards-based (runs on top of HTTP).

Advantages of REST

- REST lies with performance,with better cache support,
- lightweight requests and responses, and easier response parsing.
- REST allows for number of clients and servers, and reduces network traffic, too.
- RESTful web services are built to work best on the Web.
- Representational State Transfer (REST) is an architectural.
- In the REST architectural style, data and functionality are considered resources and are accessed using Uniform Resource Identifiers (URIs).
- The REST architectural style constrains an architecture to a client/server architecture and is designed to use a stateless communication protocol, typically HTTP.
- In the REST architecture style, clients and servers exchange representations of resources by using a standardized interface and protocol.

Principles of RESTful applications to be simple, lightweight, and fast:

- **Resource identification through URI**: A RESTful web service exposes a set of resources that identify the targets of the interaction with its clients. Resources are identified by URIs.
- **Uniform interface**: Resources are manipulated using a fixed set of four create, read, update, delete operations: PUT, GET, POST, and DELETE. PUT creates a new resource, which can be then deleted by using DELETE. GET retrieves the current state of a resource in some representation. POST transfers a new state onto a resource.
- **Self-descriptive messages**: Resources are decoupled from their representation so that their content can be accessed in a variety of formats, such as HTML, XML, plain text, PDF, JPEG, JSON, and others.
- **Stateful interactions through hyperlinks**: Every interaction with a resource is stateless; that is, request messages are self-contained. Stateful interactions are based on the concept of explicit state transfer. Several techniques exist to exchange state, such as URI rewriting, cookies, and hidden form fields

JAX-RS annotations:

Annotation	Description
@PATH(your_path)	Sets the path to base URL + /your_path. The base URL is based on your application name, the servlet and the URL pattern from the web.xml" configuration file.
@POST	Indicates that the following method will answer to a HTTP POST request
@GET	Indicates that the following method will answer to a HTTP GET request
@PUT	Indicates that the following method will answer to a HTTP PUT request
@DELETE	Indicates that the following method will answer to a HTTP DELETE request
@Produces(MediaType.TEXT_PLAIN [, more-types])	@Produces defines which MIME type is delivered by a method annotated with @GET. In the example text ("text/plain") is produced. Other examples would be "application/xml" or "application/json".
@Consumes(type [, more-types])	@Consumes defines which MIME type is consumed by this method.
@PathParam	Used to inject values from the URL into a method parameter. This way you inject for example the ID of a resource into the method to get the correct object.

Jersey Implementation

RESTful Web services that seamlessly support exposing your data in a variety of representation media types and abstract away the low-level details of the client-server communication is not an easy task without a good toolkit. In order to simplify development of RESTful Web services and their clients in Java, a standard and portable JAX-RS API has been designed. Jersey RESTful Web Services framework is open source, production quality, framework for developing RESTful Web Services in Java that provides support for JAX-RS APIs and serves as a JAX-RS (JSR 311 & JSR 339) Reference Implementation.

Jersey framework is more than the JAX-RS Reference Implementation. Jersey provides it's own API that extend the JAX-RS toolkit with additional features and utilities to further simplify RESTful service and client development. Jersey also exposes numerous extension SPIs so that developers may extend Jersey to best suit their needs.

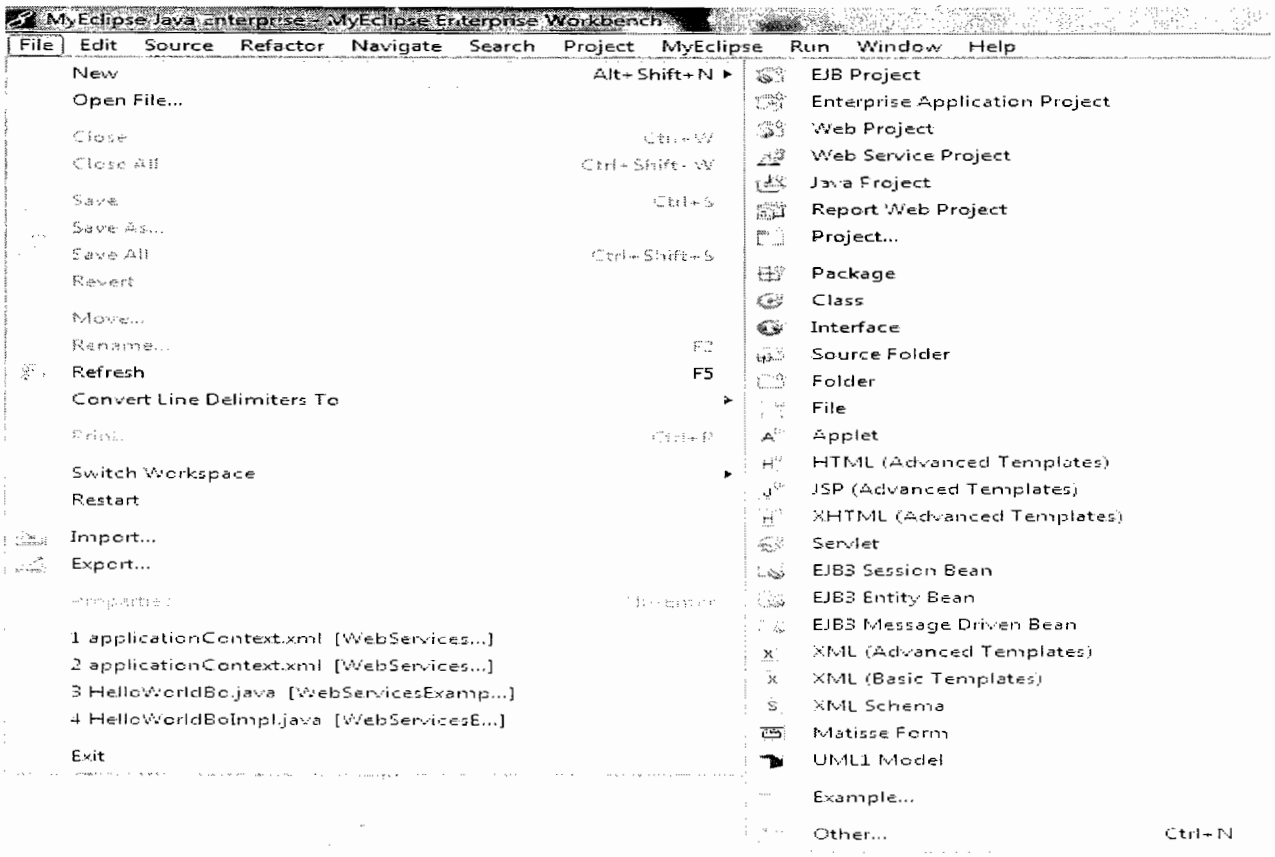
Goals of Jersey project can be summarized in the following points:

- Track the JAX-RS API and provide regular releases of production quality Reference Implementations that ships with GlassFish;
- Provide APIs to extend Jersey & Build a community of users and developers; and finally
- Make it easy to build RESTful Web services utilising Java and the Java Virtual Machine.

Example:

Using MyEclipse

Step 1:Create a web service project with the name of JAXRS



Step 2: Welcome.java

```
package edu.raju.service;
import javax.ws.rs.GET;
import javax.ws.rs.Path;
@Path("/welcome")
public class Welcome {
        @GET
        @Path("getMessage")
        public String getMessage(){
                return "Hi, Welcome to Restful Service";
        }
}
```

Step 3: web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.5"
        xmlns="http://java.sun.com/xml/ns/javaee"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
        http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd">
 <servlet>
        <display-name>JAX-RS REST Servlet</display-name>
        <servlet-name>JAX-RS REST Servlet</servlet-name>
        <servlet-class>
                com.sun.jersey.spi.container.servlet.ServletContainer
        </servlet-class>
        <load-on-startup>1</load-on-startup>
 </servlet>
 <servlet-mapping>
        <servlet-name>JAX-RS REST Servlet</servlet-name>
        <url-pattern>/services/*</url-pattern>
 </servlet-mapping>
 <welcome-file-list>
  <welcome-file>index.jsp</welcome-file>
 </welcome-file-list>
</web-app>
```

Step 4: Deploy the application in Tomcat Server and test the following URL:
http://localhost:8888/JAXRS/services/welcome/getMessage

Example 2:

```
package edu.raju.service;
import java.io.File;
import javax.ws.rs.Consumes;
import javax.ws.rs.FormParam;
import javax.ws.rs.GET;
import javax.ws.rs.POST;
import javax.ws.rs.Path;
import javax.ws.rs.PathParam;
import javax.ws.rs.Produces;
import javax.ws.rs.QueryParam;
import javax.ws.rs.core.Context;
import javax.ws.rs.core.MediaType;
import javax.ws.rs.core.Response;
import javax.ws.rs.core.UriInfo;
import javax.ws.rs.core.Response.ResponseBuilder;
import edu.raju.beans.StudentBean;
@Path("/restservice")
public class RestfulService {
        // http://localhost:8888/JAXRS/services/restservice/001/Raju/24
        @GET
        @Path("{no}/{name}/{age}")
        public String getData(@PathParam("no") int no,
                        @PathParam("name") String name, @PathParam("age") int age) {
                String details = "no:"+no+"\n name:" + name + "\n age:" + age;
                return "Details are ::" + details;
        }
        // http://localhost:8888/JAXRS/services/restservice/query?sno=10&sname=Raju
        @GET
        @Path("/query")
        public Response getDetails(@QueryParam("sno") String sno,
                        @QueryParam("sname") String sname) {
                String result = "Select * from Student where sno=" + sno + "and sname =" + sname;

                return Response.status(200).entity(result).build();
        }
        //http://localhost:8888/JAXRS/services/restservice/query?sno=10&sname=Raju
        @GET
        @Path("/context")
        public Response getDetails(@Context UriInfo info) {
                String sno = info.getQueryParameters().getFirst("sno");
                String sname = info.getQueryParameters().getFirst("sname");
                String result = "Select * from Student where sno=" + sno + "and sname =" + sname;
```

```
                return Response.status(200).entity(result).build();

        }
        // http://localhost:8888/JAXRS/services/restservice/getJson
        @GET
        @Path("/getJson")
        @Produces("application/json")
        public StudentBean getJSON() {
                StudentBean student = new StudentBean();
                student.setNo("001");
                student.setName("Raju");
                student.setAge("25");
                return student;
        }
        @GET
        @Path("/getXml")
        @Produces(MediaType.APPLICATION_XML)
        public StudentBean getXml() {
                StudentBean student = new StudentBean();
                student.setNo("001");
                student.setName("Raju");
                student.setAge("25");
                return student;
        }
        @POST
        @Path("/studentform")
        public Response addStudent(@FormParam("sno") int sno,@FormParam("sname") String
sname,@FormParam("sage") int sage){
                System.out.println("sno:"+sno+"sname:"+sname+"sage:"+sage);
                String result ="sno:"+sno+"sname:"+sname+"sage:"+sage;
                return Response.status(200).entity(result).build();
        }
         // http://localhost:8888/JAXRS/services/restservice/getImage
        @GET
        @Path("/getImage")
  @Produces("image/jpg")
  public Response getImage(){

                File file= new File("G:\\Photos\\raju.jpg");
                ResponseBuilder responseBuilder =
                                Response.ok((Object)file);
                responseBuilder.header
                     ("Content-Disposition", "attachment:" +
                                "filename=sample.jpg");
```

```
                return responseBuilder.build();
        }

        // http://localhost:8888/JAXRS/services/restservice/postJson
        @POST
        @Path("/postJson")
        @Consumes(MediaType.APPLICATION_JSON)

        public Response createTrackJSON(StudentBean bean) {
                System.out.println("createTrackJSON");
                String result = "Student Save:" + bean;
                return Response.status(201).entity(result).build();
        }
}
```

StudentBean.java

```
package edu.raju.beans;
import javax.xml.bind.annotation.XmlAttribute;
import javax.xml.bind.annotation.XmlRootElement;
@XmlRootElement
public class StudentBean {
        String no;
        String name;
        String age;
        @XmlAttribute
        public String getNo() {
                return no;
        }
        public void setNo(String no) {
                this.no = no;
        }
        public String getName() {
                return name;
        }
        public void setName(String name) {
                this.name = name;
        }
        public String getAge() {
                return age;
        }
        public void setAge(String age) {
                this.age = age;
        }
}
```

ApacheHttpClientGet.java

```
package edu.raju.test;
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import org.apache.http.HttpResponse;
import org.apache.http.client.ClientProtocolException;
import org.apache.http.client.methods.HttpGet;
import org.apache.http.impl.client.DefaultHttpClient;
public class ApacheHttpClientGet {
        public static void main(String[] args) {
                try {
                DefaultHttpClient httpClient = new DefaultHttpClient();
                HttpGet getRequest = new HttpGet(
                                "http://localhost:8888/JAXRS/services/restservice/getJson");

                        getRequest.addHeader("accept", "application/json");
                        HttpResponse response = httpClient.execute(getRequest);

                        if (response.getStatusLine().getStatusCode() != 200) {
                                throw new RuntimeException("Failed: Http error code:"
                                                + response.getStatusLine().getStatusCode());
                        }

                        BufferedReader br = new BufferedReader
                             (new InputStreamReader(response.getEntity().getContent()));
                        String output;
                        System.out.println("Out Put from Server...\n");
                        while ((output = br.readLine()) != null) {
                                System.out.println(output);
                        }
                        httpClient.getConnectionManager().shutdown();
                } catch (ClientProtocolException e) {
                        e.printStackTrace();
                } catch (IOException e) {
                        e.printStackTrace();
                }

        }
}
```

ApacheHttpClientPost.java

```
package edu.raju.test;
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import org.apache.http.HttpResponse;
import org.apache.http.client.ClientProtocolException;
import org.apache.http.client.methods.HttpPost;
import org.apache.http.entity.StringEntity;
import org.apache.http.impl.client.DefaultHttpClient;
import com.sun.org.apache.xerces.internal.util.URI.MalformedURIException;
public class ApacheHttpClientPost {
        public static void main(String[] args) {
                try {
                        DefaultHttpClient httpClient = new DefaultHttpClient();
                        HttpPost postRequest = new HttpPost(

        "http://localhost:8888/JAXRS/services/restservice/postJson");
                        StringEntity input =new
StringEntity("{\"no\":\"001\",\"name\":\"Raju\",\"age\":\"25\"}");
      input.setContentType("application/json");
                        postRequest.setEntity(input);
                        HttpResponse response = httpClient.execute(postRequest);
                        if (response.getStatusLine().getStatusCode() != 201) {
                                throw new RuntimeException("Failed: Http error code:"
                                        + response.getStatusLine().getStatusCode());
                        }
                        BufferedReader br = new BufferedReader(new InputStreamReader(
                                        response.getEntity().getContent()));
                        String output;
                        System.out.println("Out Put from Server...\n");
                        while ((output = br.readLine()) != null) {
                                System.out.println(output);
                        }
                        httpClient.getConnectionManager().shutdown();
                } catch (MalformedURIException e) {
                        e.printStackTrace();
                } catch (IOException e) {
                        e.printStackTrace();
                }

        }
}
```

JerseyClientGet.java

```
package edu.raju.test;
import com.sun.jersey.api.client.Client;
import com.sun.jersey.api.client.ClientResponse;
import com.sun.jersey.api.client.WebResource;
public class JerseyClientGet {
        public static void main(String[] args) {
                try {
                        Client client = Client.create();
                        WebResource webResource = client

        .resource("http://localhost:8888/JAXRS/services/restservice/getJson");
                        ClientResponse response = webResource.accept("application/json")
                                        .get(ClientResponse.class);
                        if (response.getStatus() != 200) {
                                throw new RuntimeException("Failed :Http error code:"
                                                + response.getStatus());
                        }
                        String output = response.getEntity(String.class);
                        System.out.println(output);
                } catch (Exception e) {
                        e.printStackTrace();
                }
        }

}
```

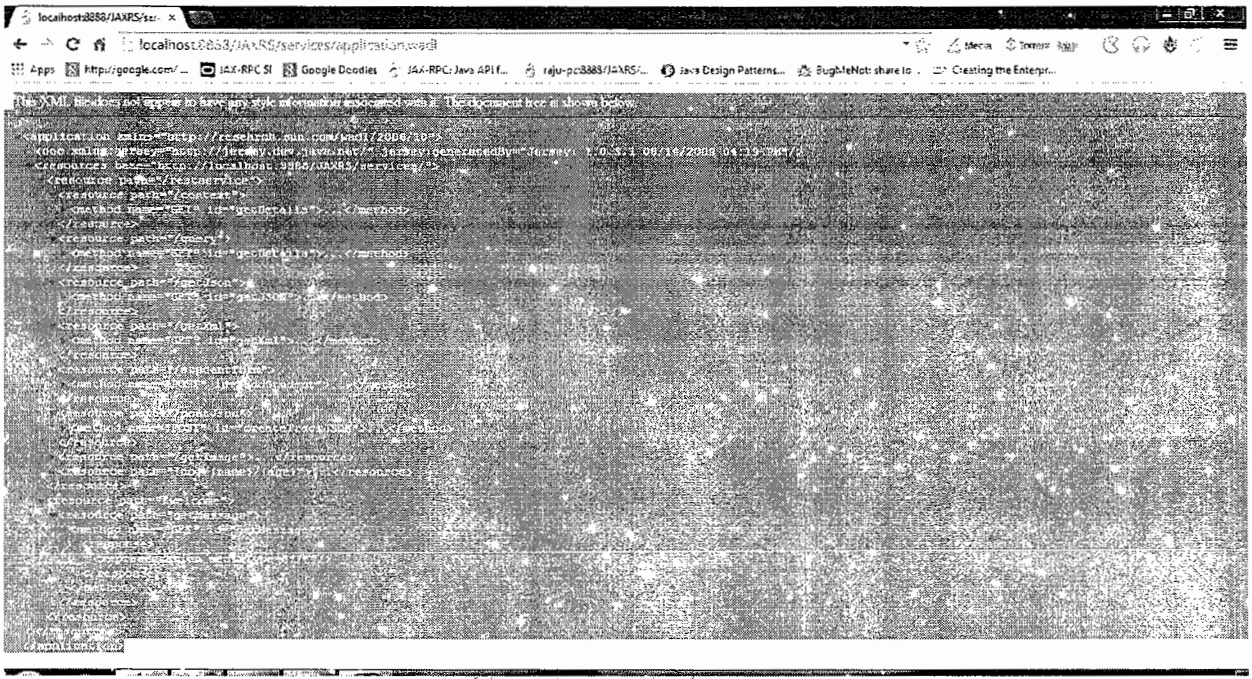
JerseyClientPost.java

```
package edu.raju.test;
import com.sun.jersey.api.client.Client;
import com.sun.jersey.api.client.ClientResponse;
import com.sun.jersey.api.client.WebResource;
public class JerseyClientPost {
        public static void main(String[] args) {
                try {
                        Client client = Client.create();
        WebResource webResource = client.resource("http://localhost: 8888/JAXRS
/services/restservice/postJson");
                        String input ="{\"no\":\"001\",\"name\":\"Raju\",\"age\":\"25\"}";
                        ClientResponse response =
webResource.accept("application/json").post(ClientResponse.class,input);
                        if (response.getStatus() != 200) {
                                throw new RuntimeException("Failed :Http error code:"
                                                + response.getStatus());
                        }
```

```
                String output = response.getEntity(String.class);
                System.out.println("O/P:"+output);
            } catch (Exception e) {
                System.out.println(e.getMessage());
                e.printStackTrace();
            }
        }
}
```

Hit the following URL for WADL file

http://localhost:8888/JAXRS/services/application.wadl



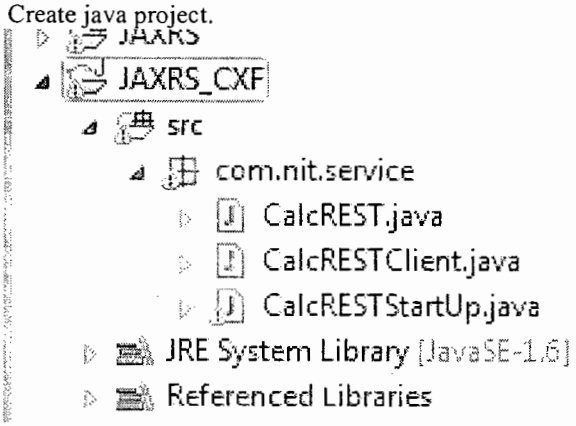
Apache CXF

Apache CXF is an open source services framework. CXF helps you build and develop services using frontend programming APIs, like JAX-WS and JAX-RS. These services can speak a variety of protocols such as SOAP, XML/HTTP, RESTful HTTP, or CORBA and work over a variety of transports such as HTTP.
For REST, CXF also supports a JAX-RS frontend.

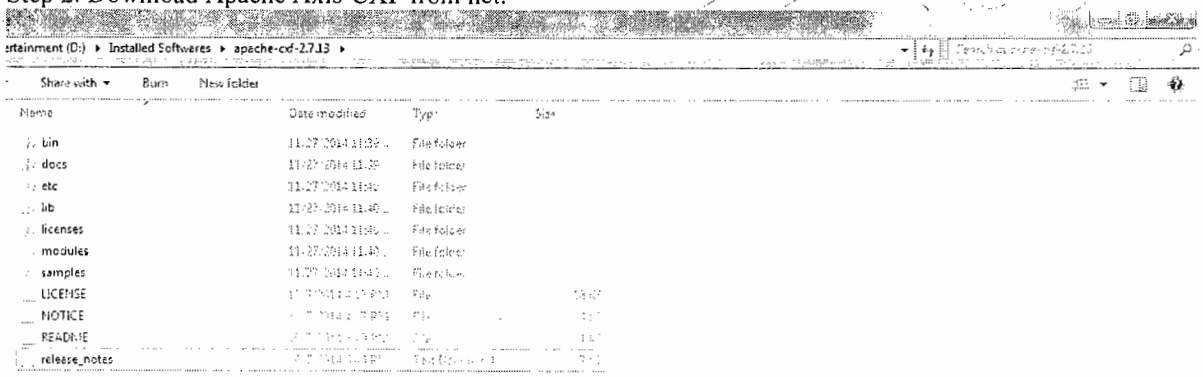
- **Ease of use**: CXF is designed to be intuitive and easy to use. There are simple APIs to quickly build code-first services, Maven plug-ins to make tooling integration easy, JAX-WS API support, Spring 2.x XML support to make configuration a snap, and much more.
- **Binary and Legacy Protocol Support:** CXF has been designed to provide a pluggable architecture that supports not only XML but also non-XML type bindings, such as JSON and CORBA, in combination with any type of transport.

Service Application:

Step 1: Create java project.



Step 2: Download Apache Axis-CXF from net.



Step 3: copy the jar files from apache-cxf lib folder into project
Step 4: Write the following files and execute it.

CalcREST.java

```
package com.nit.service;

import javax.ws.rs.GET;

import javax.ws.rs.Path;

import javax.ws.rs.PathParam;

import javax.ws.rs.Produces;

import javax.ws.rs.core.MediaType;
```

```
@Path("/calc")
public class CalcREST {

   @GET
   @Path("/add/{a}/{b}")
   @Produces(MediaType.TEXT_PLAIN)
   public String addPlainText(@PathParam("a") double a, @PathParam("b") double b) {
      return (a + b) + "";
   }

   @GET
   @Path("/add/{a}/{b}")
   @Produces(MediaType.TEXT_XML)
   public String add(@PathParam("a") double a, @PathParam("b") double b) {
      return "<?xml version=\"1.0\"?>" + "<result>" +  (a + b) + "</result>";
   }

   @GET
   @Path("/sub/{a}/{b}")
   @Produces(MediaType.TEXT_PLAIN)
   public String subPlainText(@PathParam("a") double a, @PathParam("b") double b) {
      return (a - b) + "";
   }

   @GET
   @Path("/sub/{a}/{b}")
   @Produces(MediaType.TEXT_XML)
   public String sub(@PathParam("a") double a, @PathParam("b") double b) {
      return "<?xml version=\"1.0\"?>" + "<result>" +  (a - b) + "</result>";
   }
}
```

CalcRESTStartUp.java

```
package com.nit.service;
import org.apache.cxf.endpoint.Server;
import org.apache.cxf.jaxrs.JAXRSServerFactoryBean;
import org.apache.cxf.jaxrs.lifecycle.SingletonResourceProvider;
public class CalcRESTStartUp {
   public static void main(String[] args) {
        JAXRSServerFactoryBean sf = new JAXRSServerFactoryBean();
        sf.setResourceClasses(CalcREST.class);
        sf.setResourceProvider(CalcREST.class,
           new SingletonResourceProvider(new CalcREST()));
```

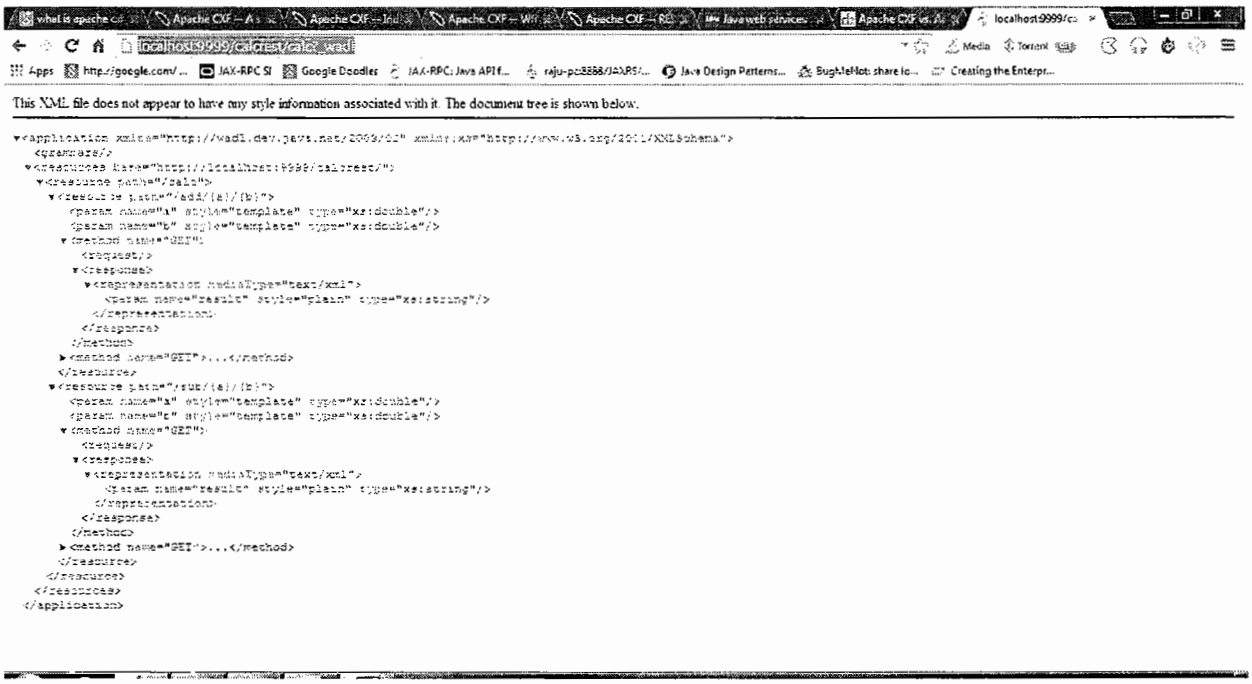
```
        sf.setAddress("http://localhost:9999/calcrest/");
        Server server = sf.create();
        // destroy the server
        // uncomment when you want to close/destroy it
       // server.destroy();
   }
}
```

Client Application:

Step 1: Write java file to execute the client.

```
package com.nit.service;
import org.apache.cxf.jaxrs.client.WebClient;
public class CalcRESTClient {
   static final String REST_URI = "http://localhost:9999/calcrest/";
   static final String ADD_PATH = "calc/add";
   static final String SUB_PATH = "calc/sub";

   public static void main(String[] args) {
      int a = 122;
      int b = 34;
      String s = "";

      WebClient plainAddClient = WebClient.create(REST_URI);
      plainAddClient.path(ADD_PATH).path(a + "/" + b).accept("text/plain");
      s = plainAddClient.get(String.class);
      System.out.println(s);

      WebClient xmlAddClient = WebClient.create(REST_URI);
      xmlAddClient.path(ADD_PATH).path(a + "/" + b).accept("text/xml");
      s = xmlAddClient.get(String.class);
      System.out.println(s);

      WebClient plainSubClient = WebClient.create(REST_URI);
      plainSubClient.path(SUB_PATH).path(a + "/" + b).accept("text/plain");
      s = plainSubClient.get(String.class);
      System.out.println(s);

      WebClient xmlSubClient = WebClient.create(REST_URI);
      xmlSubClient.path(SUB_PATH).path(a + "/" + b).accept("text/xml");
      s = xmlSubClient.get(String.class);
      System.out.println(s);
   }
}
```

Step 2: hit the following URL for WADL file.
http://localhost:9999/calcrest/calc?_wadl



Important Questions.

Distributed Application:

1. What is Distributed Application?
2. What is Distributed Technology?
3. What are the benefits of Distributed Application and briefly explain?
4. How to develop distributed applications in JAVA?
5. What are the difference between CORBA, Socket Programming, RMI,EJB?

Web services.

6. What is the need of Web Service?
7. Can you please explain real-time use of Web Service?
8. What are the primary components of Web Service?
9. What do mean by Inter-operability, briefly explain it?
10. Explain the architecture of Web Service?

What is SOA?

11. What is the principle of SOA?
12. How SOA applications are related to Web Service?
13. What are the components of SOA?

Web Service Roles

14. Briefly explain the functionality Service Provider?
15. Briefly explain the functionality Service Requestor?
16. Briefly explain the functionality Service Registry?
17. What are primary roles of Service Provider, Service Requester, Service Registry?

Web services Architecture

18. What is an architecture of Web Service?
19. What are the components of Web Service?
20. Briefly explain the functionality of Provider.Consumer,Registry in Web Service?

WS-I (Web Service Interoperability) Organization.

21. What is WS-I?
22. What do you mean by Interoperability?

Web Services Development Parts.

23. Briefly explain WS-I specifications?
24. Difference between BP-1.0 and BP1.1 specifications?
25. What are the API's to implement We Service?
26. What are the implementations of Web Services in Java?
27. Briefly explain the implementation approaches of Web Service?
28. Difference between Contract-First and Contract-Last i.e (Top-down and Bottom-up) approach?
29. Which is suitable and benefits of Top-down and Bottom-up approches?
30. What do mean by end-point?
31. Difference between servlet-endpoint and ejb-end point applications?
32. What are the Message Exchange Patterns(MEP) are there?
33. Difference between synchronous and asynchronous applications?
34. What are the Message Exchange Formats?
35. Briefly explain the rpc-encoded,rpc-literal, document-encoded,
36. What is style and use in message exchange formats?

WSDL

37. Define WSDL and purpose?
38. What is the need of WSDL?
39. Define the elements in WSDL?

40. Briefly explain the functionality of Definition element in WSDL?
41. Briefly explain the functionality of Types element in WSDL?
42. Briefly explain the functionality of Messages element in WSDL?
43. Briefly explain the functionality of Operation element in WSDL?
44. Briefly explain the functionality of Port-Type element in WSDL?
45. Briefly explain the functionality of Binding element in WSDL?
46. Briefly explain the functionality of Service element in WSDL?
47. Briefly explain the functionality of Port element in WSDL?

SOAP

48. What is SOAP?
49. What is the usage of SOAP in web service?
50. Difference between SOAP and HTTP protocal?
51. What are the elements in SOAP?
52. Define SOAP Header, SOAP body, SOAP fault?
53. What are messaging styles of SOAP?
54. What are the encoding formats of SOAP?
55. What is SOAP binding?
56. Difference between SOAP binding and HTTP Binding?

UDDI

57. What is UDDI?
58. What is the functionality of UDDI?
59. How to secure the data in UDDI registry?
60. How to publish and discovery the services from UDDI?
61. What does UDDI registry contains?

JAX-RPC, JAX-RPC-SI web service development

62. What is JAX-RPC?
63. What are the implementations of JAX-RPC?
64. Different approaches of web services using JAX-RPC?
65. Difference between different implementations of JAX-RPC?
66. Explain briefly about JAX-RPC SI, AXIS 1, Web logic ,Web sphere implementations?
67. What is an SEI (Service End Point) interface in JAX-RPC?
68. What the rules to write an interface in JAX-RPC?
69. Different approaches to write consumer side applications?
70. Briefly explain Stub-based consumer?
71. Briefly explain Dynamic-Proxy based consumer?
72. Briefly explain Dynamic-Invocation Interface consumer?

JAX-RPC Handlers

73. What is Handlers?
74. What is the basic functionality in handlers?
75. How to implement consumer side handlers?
76. How to implement provider side handlers?

JAX-WS

77. What is JAX-WS?
78. What are the implementations of JAX-WS?
79. Explain briefly about JAX-RPC RI, Metro, AXIS 2, Web logic ,Web sphere implementations?
80. What is an SEI (Service End Point) interface in JAX-WS?
81. What the rules to write an interface in JAX-WS?
82. Difference between JAX-RPC and JAX-WS?
83. Which is the best one JAX-RPC or JAX-WS?

JAX-WS Handlers

84. How to implement the Handlers in JAX-WS?
85. Difference between Handlers in JAX-RPC and JAX-WS?

WS-SECURITY

86. What is WS-Security?
87. Briefly explain about WS-Security?
88. Briefly explain about WS-Security using Encryption?
89. Briefly explain about WS-Security using Public, Private Key?
90. Briefly explain about WS-Security using Digital Signatures?
91. Briefly explain about WS-Security using SAML?
92. Briefly explain about WS-Security using SSL?

WS-* Standards.

93. What are the WS-I standards?

Interaction with Dot Net(.Net) Framework

94. How to interact with .Net framework?
95. Shall we implement Java as provider and Consumer as .Net and Vice versa?

Tools & IDE's Used for XML and Web services

96. What are the IDE's are using to work with Webservice?
97. Explain the need of Altova XML Spy?
98. Explain the need of SOAP UI?
99. How to test web services using SOAP UI?
100. How to write test cases using SOAP UI?
101. Explain the need of TCP IP Monitor?

Restful Web service

102. What is REST and RESTful web services?
103. What is Representational State Transfer?
104. Differences between SOAP oriented and Restful Web services?

JAX-RS

105. What is JAX-RS?
106. How to implement Restful Web services?
107. What are the different implementations of JAX-RS?
108. Briefly explain about Jersey,Restlet, RestEasy,Apache CXF?
109. What are the Self-Descriptive messages in restful services?
110. What is Web Resource?
111. How to modify,delete,read web resources?
112. What are the Http methods used to modify, delete, read(CRUD) operations from web resources?
113. How to handle the exceptions in JAX-RS?
114. What is the difference between HTTP POST and PUT requests in REST ?
115. What are the different formats supported by REST API?
116. What are the real world examples of REST Web Service ?
117. How can you implement security in Restful services?

----------THE END----------

Thank You

Have A Nice Day