Developing Java Web Services

Session: 1

Introduction to Web Services

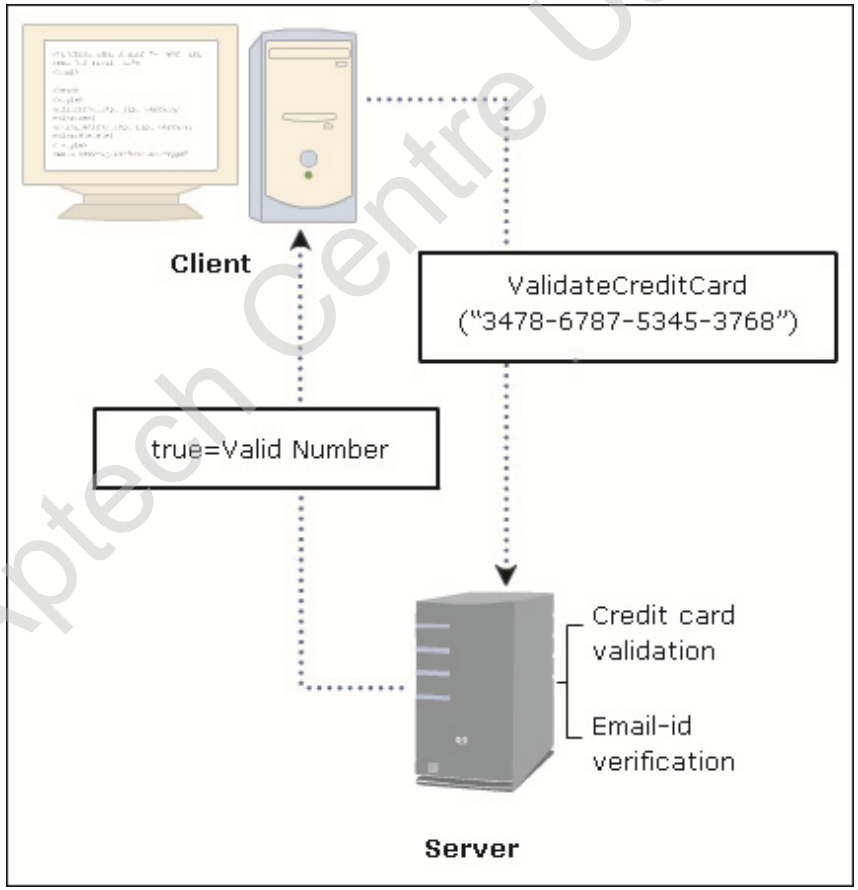
Objectives

- Define Web services and describe their purpose
- Describe Service Oriented Architecture (SOA)
- Describe role of XML, SOAP, Registry standards, and WSDL in Web services
- Describe purpose of using JAXP in processing XML documents
- Explain parsing of XML document using SAX and DOM
- Describe JAX-WS
- Explain JAXR architecture, its components, and its interfaces
- Describe the purpose and features of SAAJ
- Describe purpose and limitations of JAXB and its components
- Explain the marshalling and unmarshalling processes

Introduction to Web Services

A Web service is a service available on the Web. In other words, they are methods made available on the Web.

- Following figure shows the client-server architecture:



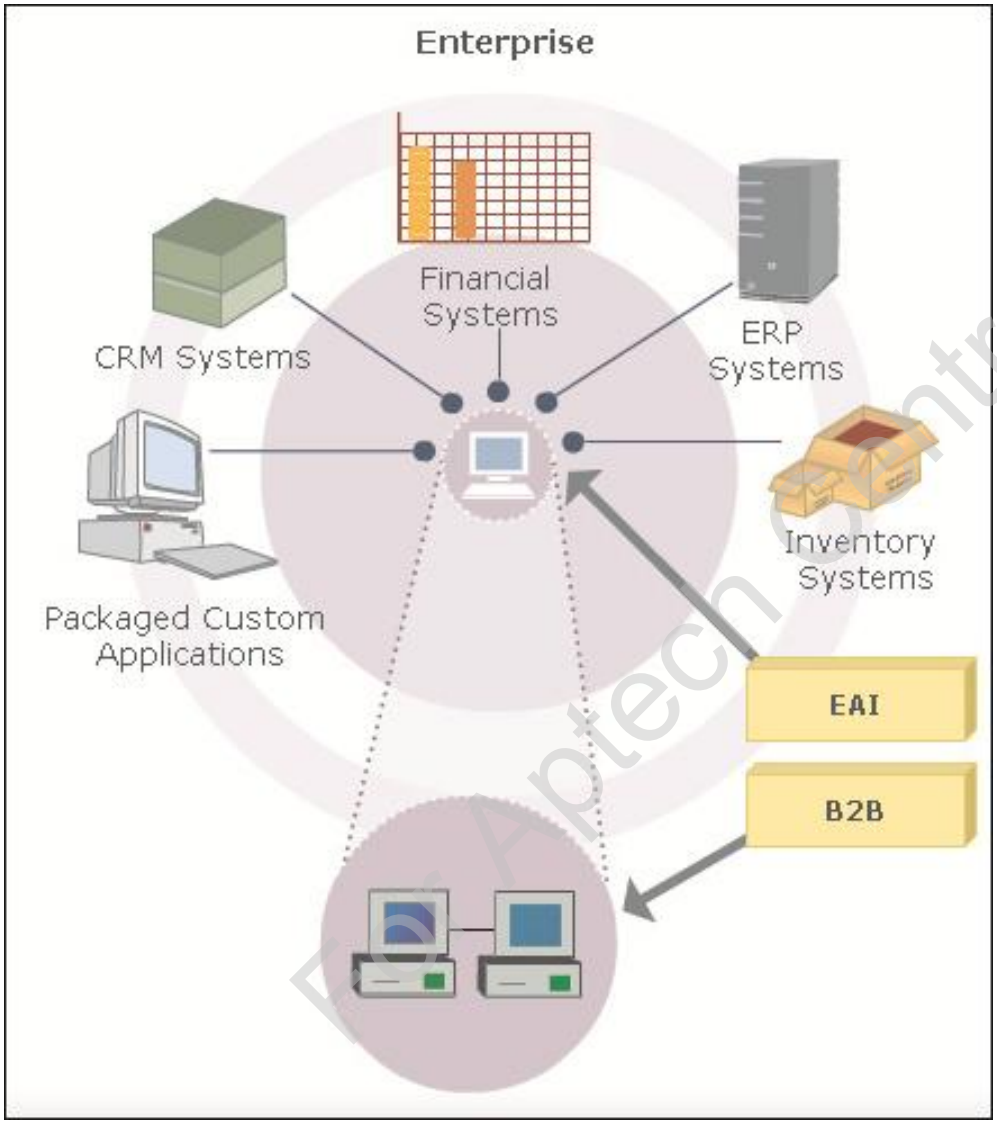
Characteristics of Web Services

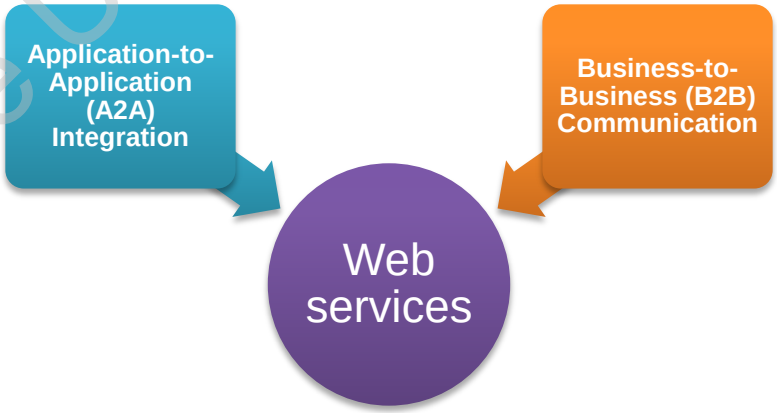
- Characteristics of Web services are as follows:

Accessibility	• A Web service is accessible over the Web.
Communication Standards	• Standard Web protocols interact with other Web services or application programs using XML to exchange information.
Integration	• Web services are loosely coupled and integrated only when required.

Uses of Web Services

- Following figure shows the uses of Web services:





Advantages of Web Services

Freedom to use any platform and language for code development

- Developers can use Web services on any software platform, architecture, and programming language for development.

Flexibility to reuse existing software code

- Web services allow enterprise application developers to use and reuse software code.

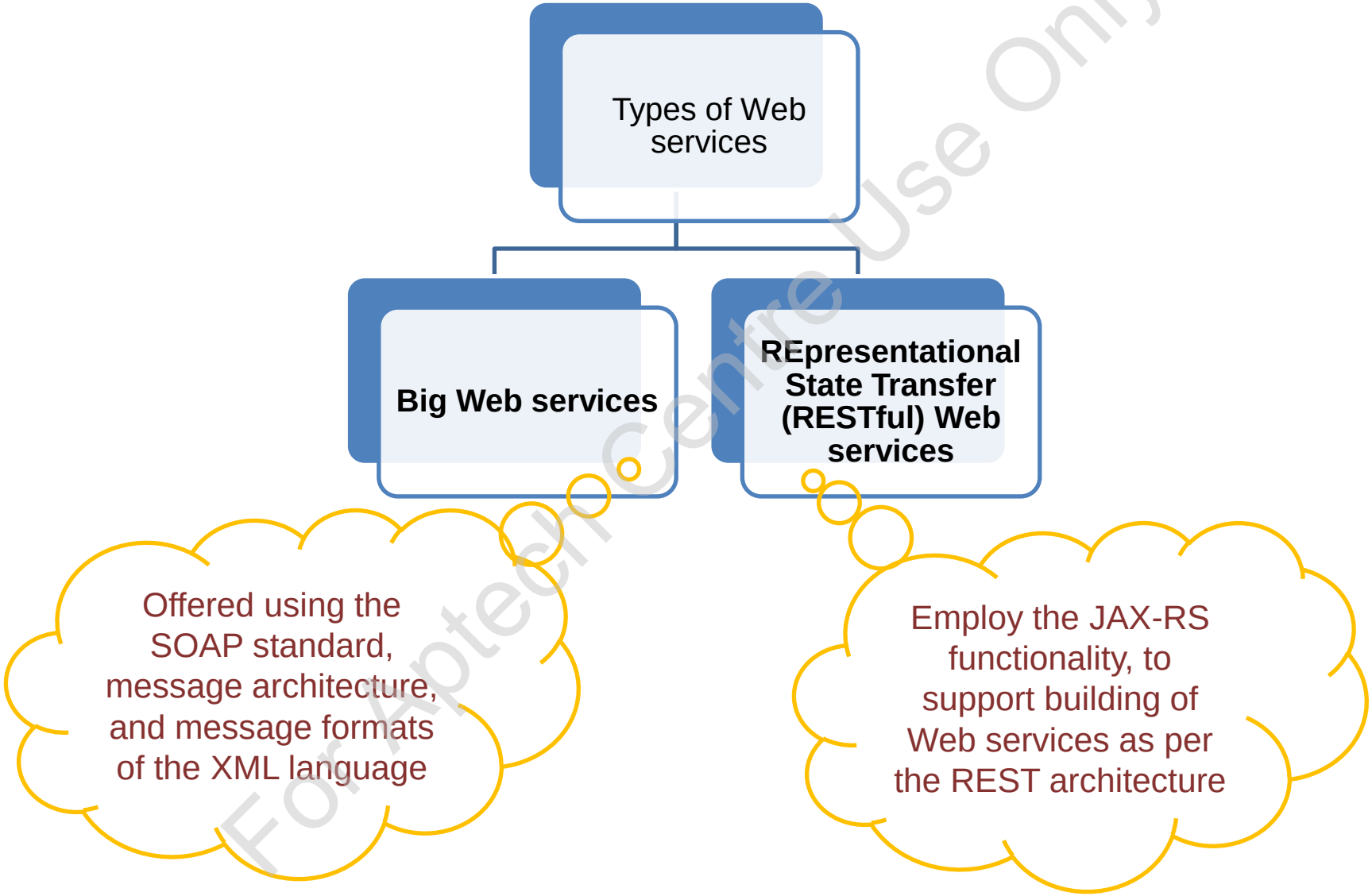
Availability of wide variety of tools

- A large variety of tools is available for developing Web services.

Wider market for services

- Extending applications and services as Web services avail organizations a vast range of clientele.

Types of Web Services



Service Oriented Architecture (SOA)

- Service Oriented Architecture (SOA) involves building an infrastructure that provides location and implementation transparency.
- Following table lists the components of a typical SOA:

Component	Description
Service Broker	Publishes information related organizations and their services and provides information on the accessibility and usage of the services provided.
Service Provider	Allows service consumers to use the services depending on the service cost, availability of the service, and security.
Service Consumer	Is used in an organization internally/externally. It makes use of a service broker to locate one or more services. Then, it connects to the service provider to make use of services.

SOA Implementation by Web Services 1-2

Web service is created by service provider and hosted on the server. It uses XML to locate and implement transparency.

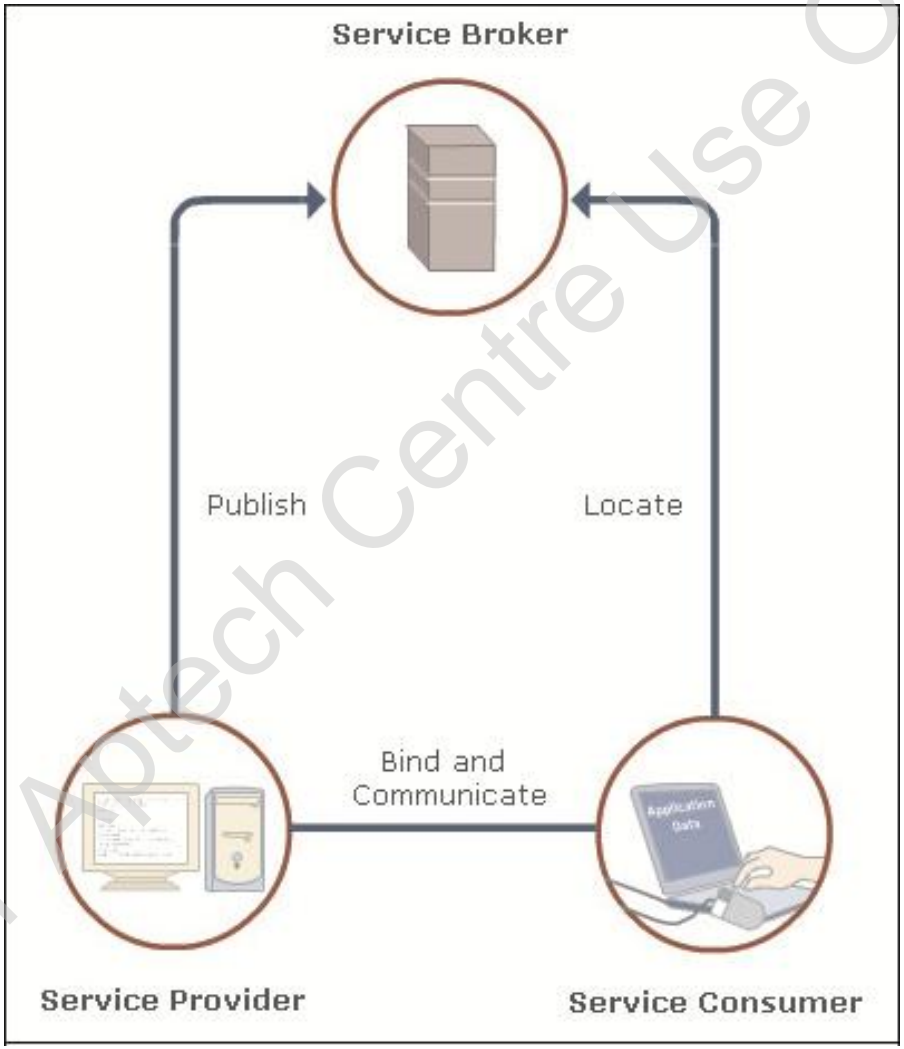
The information related to accessibility and usage is published in registries such as UDDI or ebXML, which serve as service brokers.

The service consumer identifies the required service and gets its information. Based on this information, the service consumer invokes the Web service that is placed on the service provider's server.

XML is used for communication and Hypertext Transfer Protocol (HTTP) is used as the communication protocol.

SOA Implementation by Web Services 2-2

- Following figure illustrates the registry:



Web Service Standards

- Web service standards include the following:
  - Exchange information over the Web
  - Transfer data
  - Maintain registries
  - Ensure interoperability between different platforms

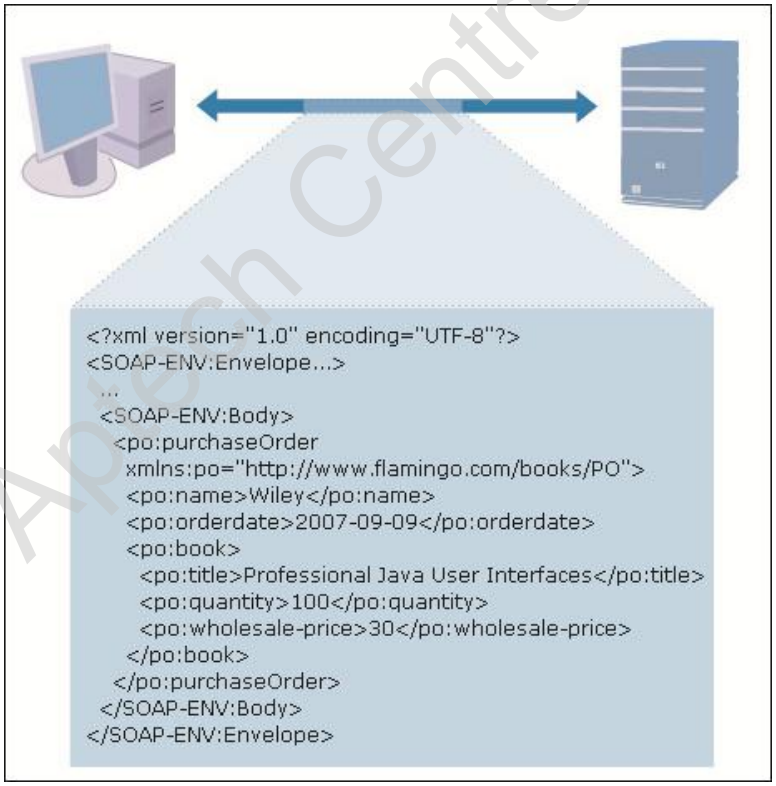
Extensible Markup Language (XML)

- The most common markup language to communicate over the Web.
- The most flexible language that allows applications to communicate irrespective of their geographic locations or software platforms.
- Helps to implement and execute Web services.

Simple Object Access Protocol

Simple Object Access Protocol (SOAP) is a standard protocol that facilitates transfer of XML data among various applications.

- SOAP message (SOAP envelope) is an XML document.
- SOAP message has a header and a body that has message data enclosed in an envelope.
- Following figure shows the structure of SOAP message:

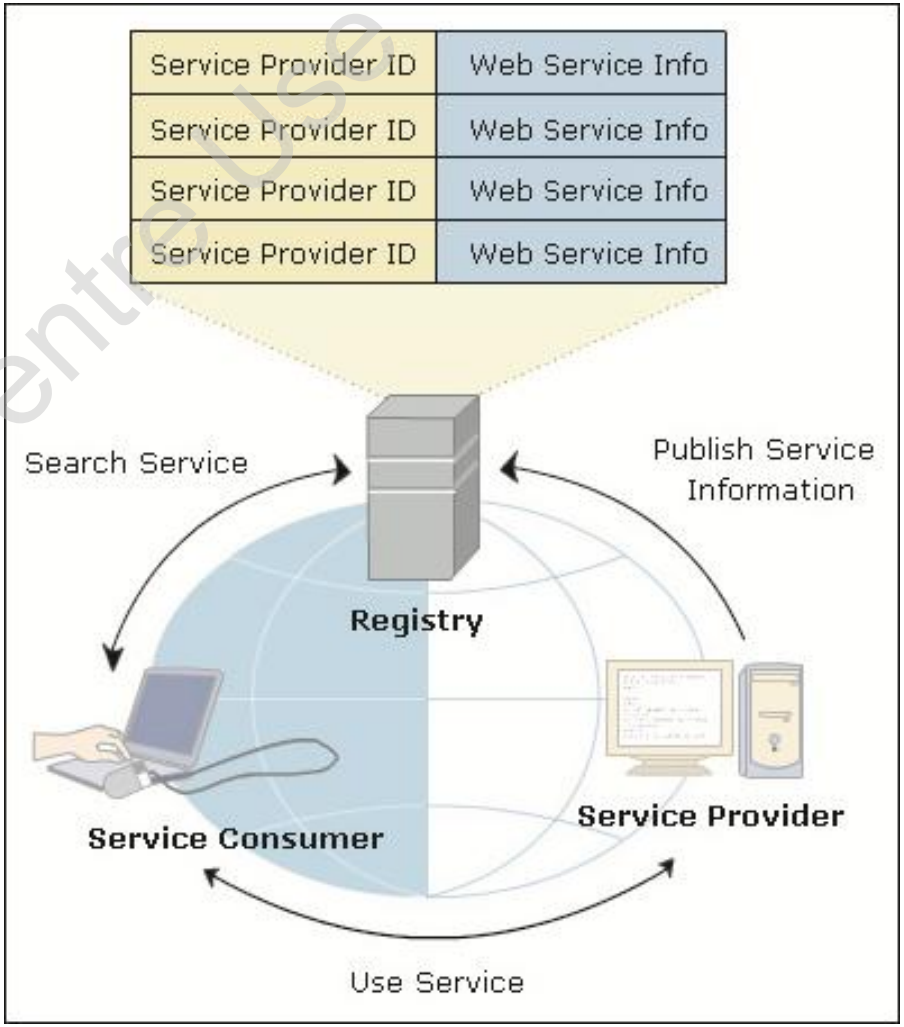


Web Service Registries

- Every service provider is listed in the registry.
- Following figure depicts the Web service registries:

Web service registry is a service that allows service providers to publish services on the Web.

First, create a registry account and add information including name and service of service provider, Web service name, type of service, and contact information of the service provider.



Web Service Description Language

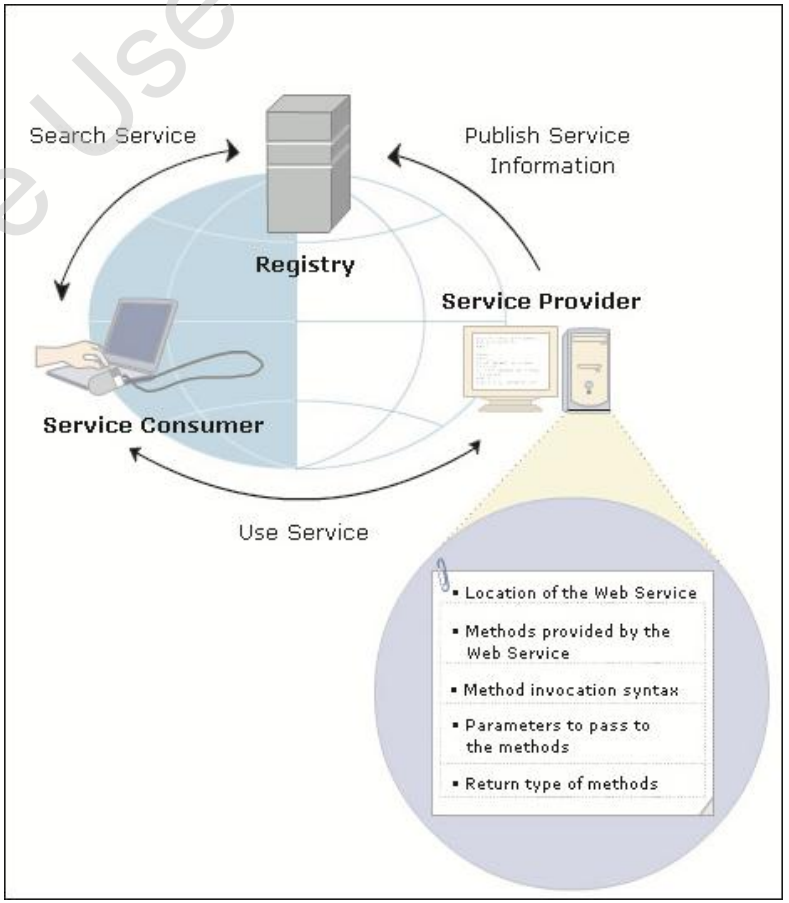
- Web services can be used by first retrieving a document (written in WSDL) from the registry.

WSDL Document Information

- Web service location that refers to the WSDL file
- Parameters that have to be passed to the methods
- Methods provided by the Web service
- Syntax to invoke method
- Return type of methods

- WSDL need not be coded manually as Integrated Development Environments (IDEs) automatically generate the document.

- Following figure depicts the location of WSDL file:

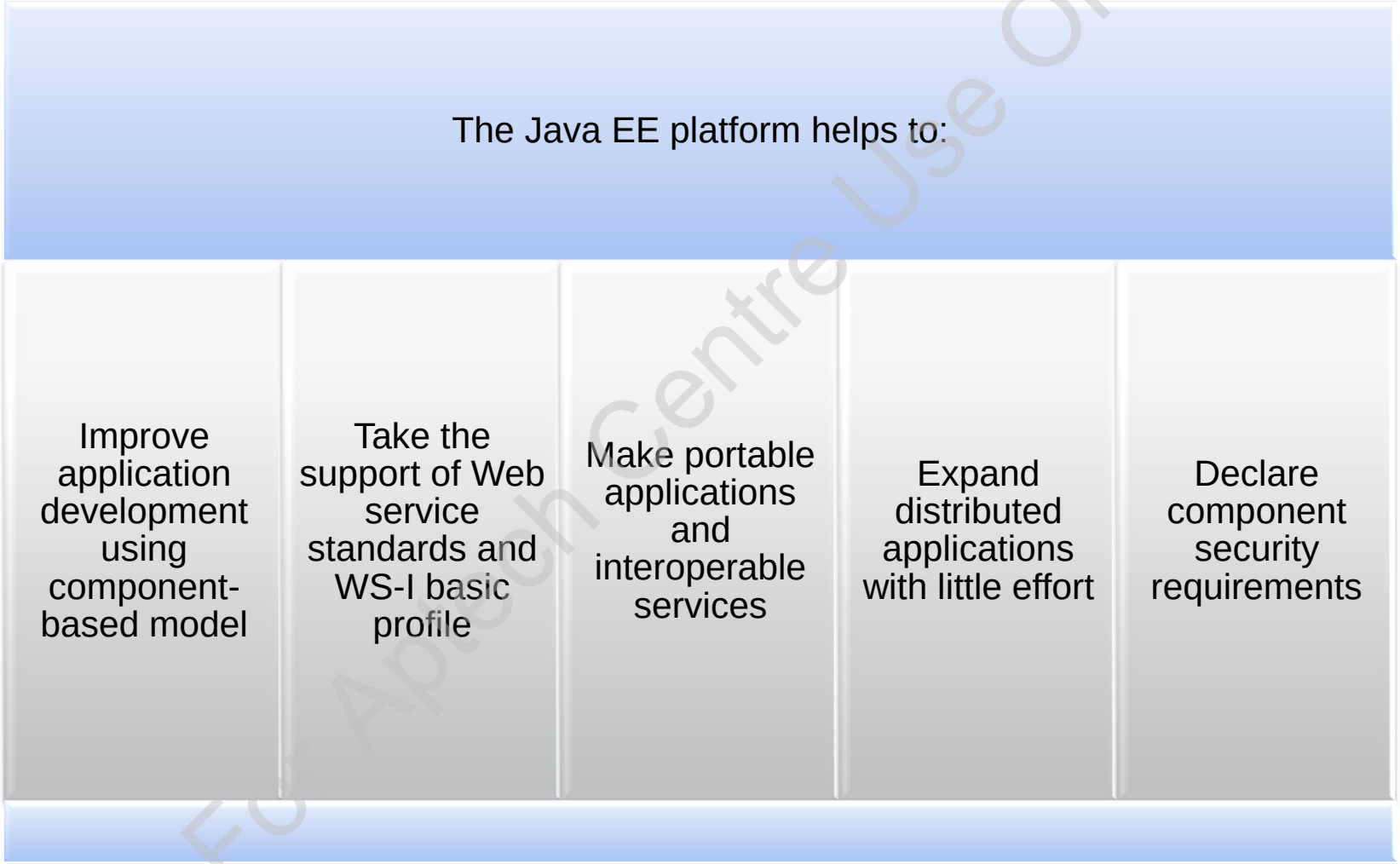


Java EE 7 Web Services Framework

- Following table describes the four Web service APIs in Java EE that implement Web service applications:

Web Service API	Description
Java API for XMLbased Web services (JAX-WS)	It enables communication with Java and non-Java Web services.
SOAP with Attachments API for Java (SAAJ)	It helps read, manipulate, create and transmit SOAP messages, and process SOAP header blocks in JAX-RPC. It complies with SOAP 1.1 and the SOAP Messages with Attachments specification.
Java API for XML Registries (JAXR)	It allows an application to access registries such as UDDI or ebXML. It simplifies publishing and querying of Web services.
Java API for XML Processing (JAXP)	It enables users to use DOM2 and SAX2 interfaces. Apart from this API, standard Java APIs can also read, write, and modify XML documents.

Benefits of Using the Java EE Platform



Web Services Technologies of Java EE 7

- Following table lists the Web Services Technologies of Java EE 7:

Web Service Technology	JSR	Description
Java API for RESTful Web Services (JAX-RS) 2.0	339	Assists in the creation of an API that supports RESTful Web services in the Java Platform.
Implementing Enterprise Web Services 1.3	109	Defines the programming model and runtime architecture to implement Web services in Java.
Java API for XML-based Web Services (JAX-WS) 2.2	224	Is the next generation Web services API replacing JAX-RPC 1.0.
Web Services Metadata for the Java Platform	181	Defines an annotated Java format that uses Java Language Metadata (JSR 175).
Java API for XML-based RPC (JAX-RPC) 1.1 (Optional)	101	Supports emerging industry XML- based RPC standards.
Java APIs for XML Messaging 1.3	67	Helps to package and transport business transactions. It uses on-the- wire protocols defined by ebXML.org, Oasis, W3C, and IETF.
Java API for XML Registries (JAXR) 1.0	93	Assists a set of distributed Registry Services to enable business-to-business integration between enterprises. It uses the protocols defined by ebXML.org, Oasis, and ISO 11179.

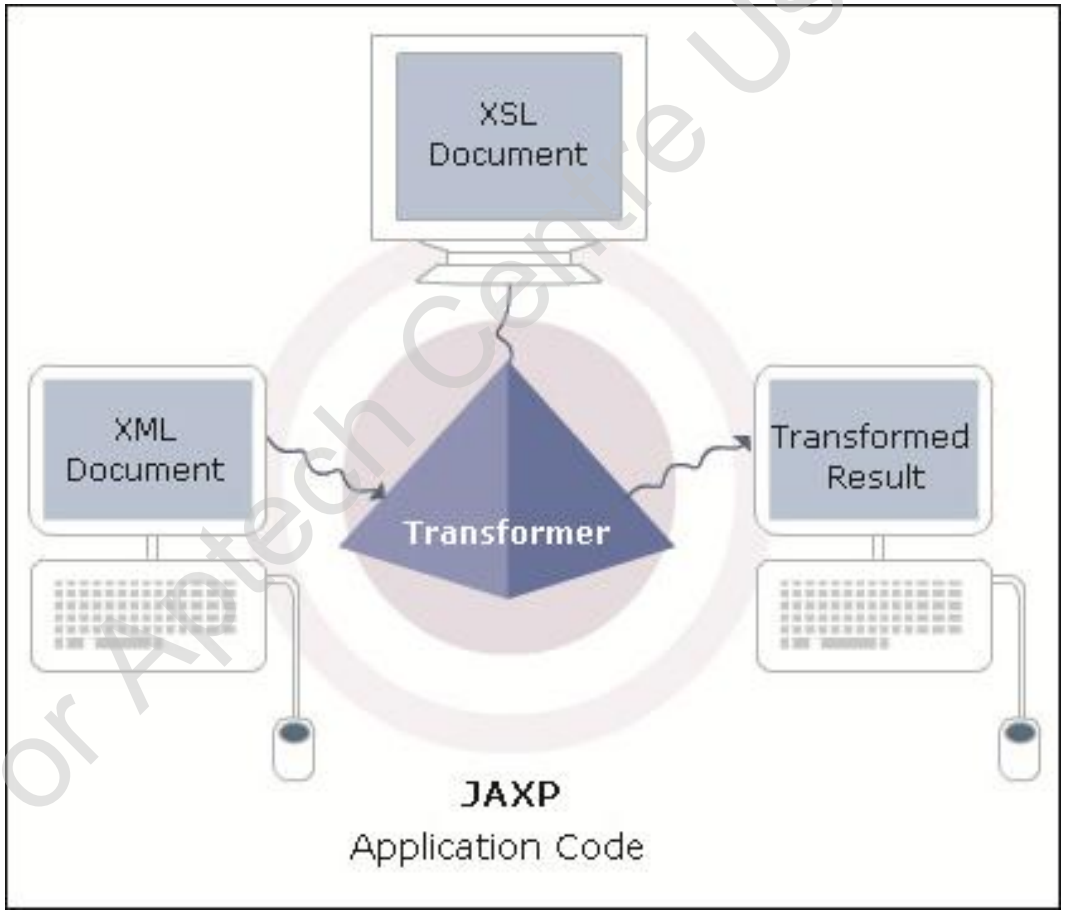
Implementation of Web Service Using Java Class

- Two types of endpoints are: JAX-WS endpoint and EJB endpoint in Java EE Web services.
- Steps to develop a JAX-WS-based endpoint are as follows:

Defining remote interface	• A remote interface is defined, which has all methods as a part of the service that would be exposed in the service. • These methods throw java.rmi.RemoteException. This interface also extends java.rmi.Remote interface.
Implementing remote interface	• The class that implements the interface is defined. • The class has all the methods declared in the remote interface.
Building service	• The .java files of remote interface and implementation class are compiled. • The WSDL file and mapping file are created using the wscompile tool.
Packaging and deployment	• The class files, WSDL file and mapping file are packaged into a WAR file and deployed on a server.

JAXP 1-2

- JAXP uses Extensible Stylesheet Language Transformation (XSLT) engines to transform XML documents from one format to another.
- Following figure shows processing of XML documents using JAXP:



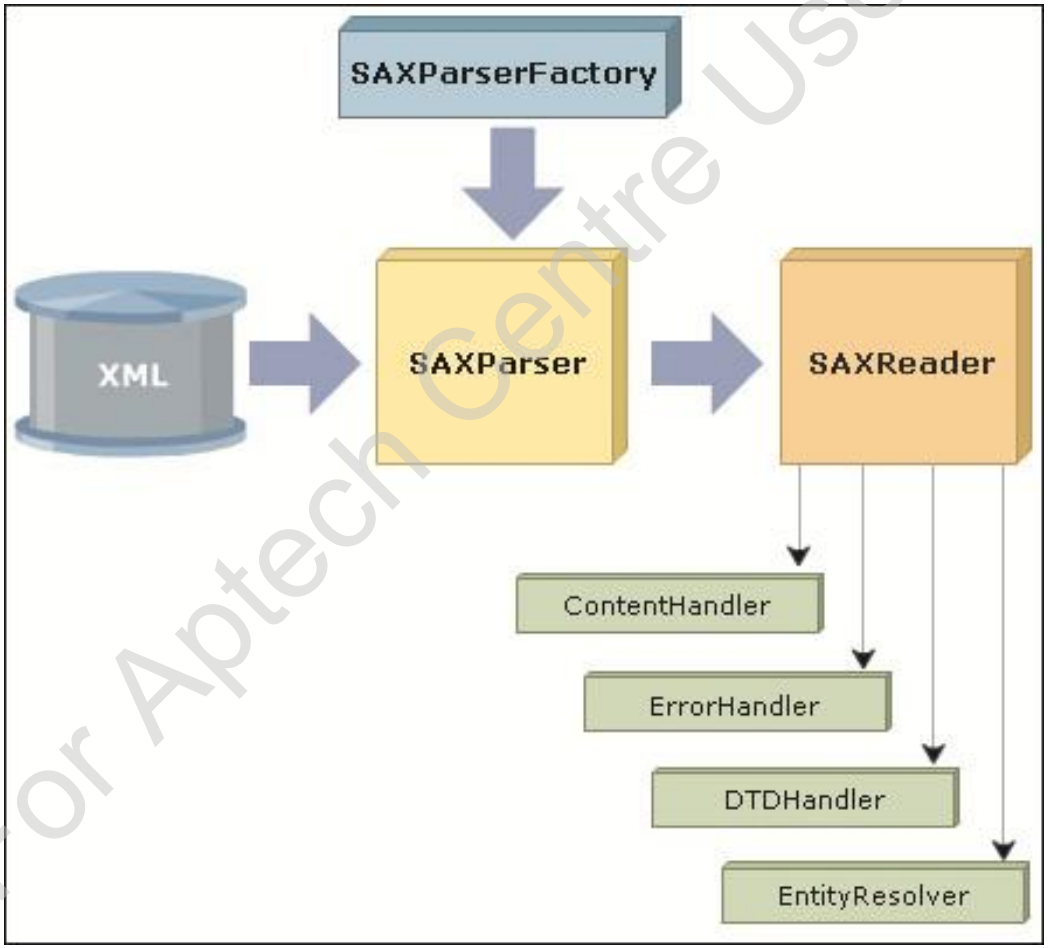
JAXP 2-2

- Following table lists the packages that define the JAX API:

Package	Description
javax.xml.parsers	Defines SAXParserFactory and DocumentBuilderFactory classes, which provide a common interface for SAX and DOM parsers.
javax.xml.transform	Defines the generic APIs to transform source to result.
javax.xml.transform.dom	Implements DOM-specific transformation APIs.
javax.xml.transform.sax	Implements SAX2-specific transformation APIs.

SAX Parser 1-2

- SAX processes the XML documents sequentially into a series of events.
- SAX is very fast as it does not load the document into memory.
- The SAX parser is depicted in the following figure:



SAX Parser 2-2

- Following table lists the key classes/interfaces to parse XML documents:

Package	Description
DefaultHandler	Is present in the org.xml.sax.helpers package. It acts as a default base class for SAX2 event handlers.
SAXParserFactory	Is present in the javax.xml.parsers package. It returns the SAXParser and the exception classes to report errors.
SAXParser	Is present in the javax.xml.parsers. It defines the API that wraps an XMLReader implementation class.
XMLReader	Is present in the java.io package. It is an interface to read an XML document by using callbacks.

SAX Parser Implementation 1-2

- Parsing an XML document is done with the help of JAXP APIs and SAX.
- Following Code Snippet shows how to parse an XML document using a SAX-based parser:

```
//Step 1
public class SAXParsing extends DefaultHandler{

        public void readDocument(){
        //Step 2
        SAXParserFactory spFactory = SAXParserFactory.newInstance();
        spFactory.setValidating (true);
        //Step 3
        SAXParser saxp = spFactory.newSAXParser();
        //Step 4
        XMLReader xReader = saxp.getXMLReader();
        //Step 5
        xReader.setContentHandler(this);
        //Step 6
        xReader.parse(XMLDocument);
    }
}
```

SAX Parser Implementation 2-2

- Steps performed by the code are as follows:

1. A class named SAXParsing is created. This class extends the DefaultHandler class.
2. An instance of the SAXParserFactory class, named spFactory, is created, and the validation property of this instance is set to true.
3. An instance of the SAXParser class, named saxp, is obtained from the SAXParserFactory instance.
4. The encapsulated SAX XMLReader is retrieved to read character streams.
5. The ContentHandler property of XMLReader is set to allow the application to register a content event handler.
6. The XML document is parsed using the instance of the XMLReader interface.

DOM Parser 1-3

A DOM parser accesses XML documents randomly and creates a tree from its elements. It splits the documents into more memory, but is easy to create and modify XML documents by using in-memory content tree.

- Following table lists the packages in DOM parser:

Package	Description
org.w3c.dom	Defines DOM programming interfaces for XML documents. Its primary interfaces are Document and Node. Document is a HTML/XML document and Node is the primary data type for DOM.
javax.xml.parsers	Defines the DocumentBuilder class and DocumentBuilderFactory class to process XML documents.

- Parsing an XML document can also be done with the help of JAXP APIs and DOM.

DOM Parser 2-3

- Following Code Snippet shows how to parse an XML document using DOM-based parser:

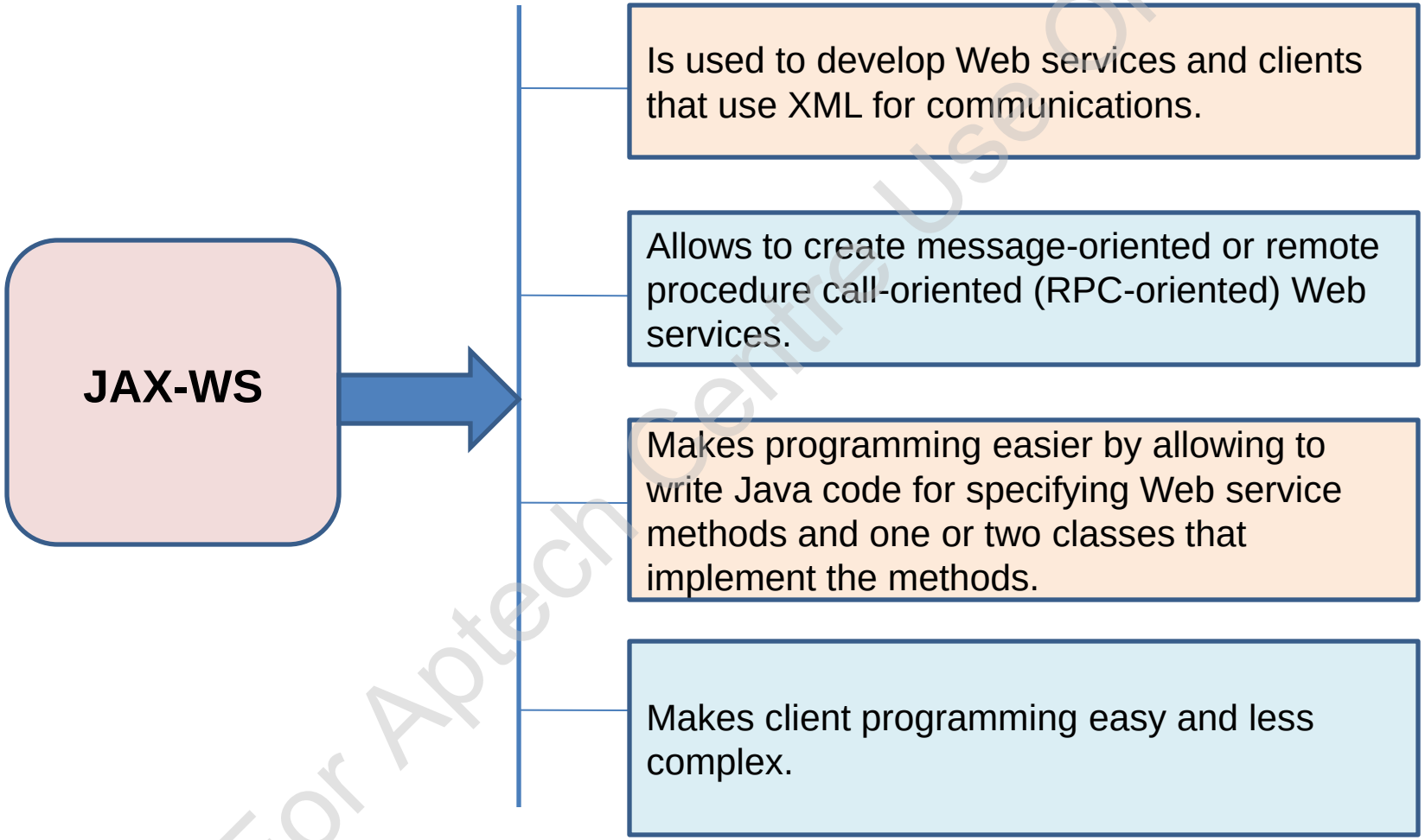
```
. . .
public void readDocument(){
// step 1
DocumentBuilderFactory dbFactory = DocumentBuilderFactory. newInstance();
dbFactory.setvalidating(true);
// step 2
DocumentBuilder dBuilder = dbFactory.newDocumentBuilder();
// step 3
Document dcm = dBuilder.parse(XMLDocument);
// parse the tree created - node by node
 }
}
```

DOM Parser 3-3

- Steps performed by the code are as follows:

1. A class named DOMParsing is created for parsing an XML document. An instance of the DocumentBuilderFactory class named dbFactory is created and the validation property of this instance is set to true.
2. A DocumentBuilder object named dBuilder is created using an instance of DocumentBuilderFactory class.
3. The instance of the DocumentBuilder class parses the input file by invoking the parse method and passing the document to be parsed named XMLDocument.

JAX-WS

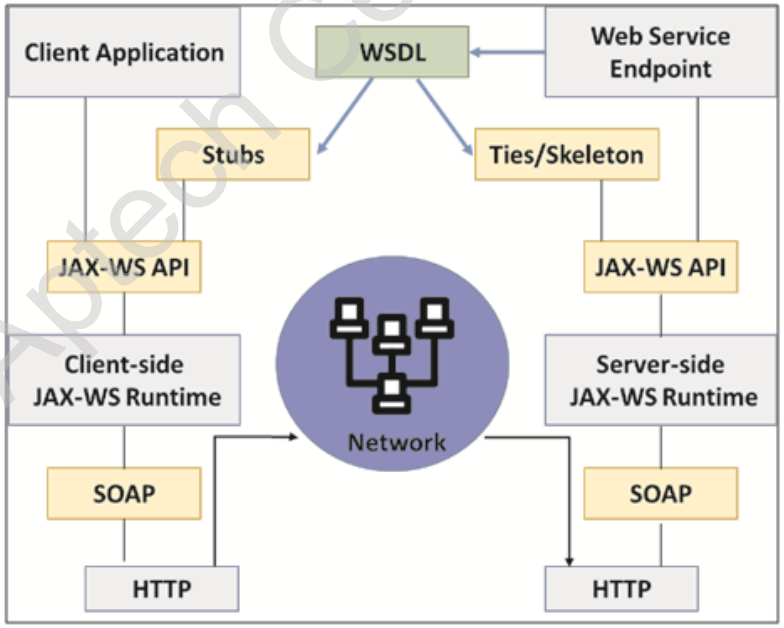


Role of JAX-WS in Web Services

Tasks performed by JAX-WS in a Web service are as follows:

Interoperates with SOAP-based Web services using WSDL	Maps the data types between XML and Java	Invokes methods on the generated stubs	Supports standard Internet protocols such as HTTP	Enables portability of service endpoints and service clients across JAX-WS implementations

- Following figure shows the role of JAX-WS in Web services:



Features of JAX-WS

Enables interoperability with any SOAP-based Web services using WSDL.

Provides mapping of data types between XML and Java.

Invokes methods on the generated stubs.

Supports standard Internet protocols such as HTTP.

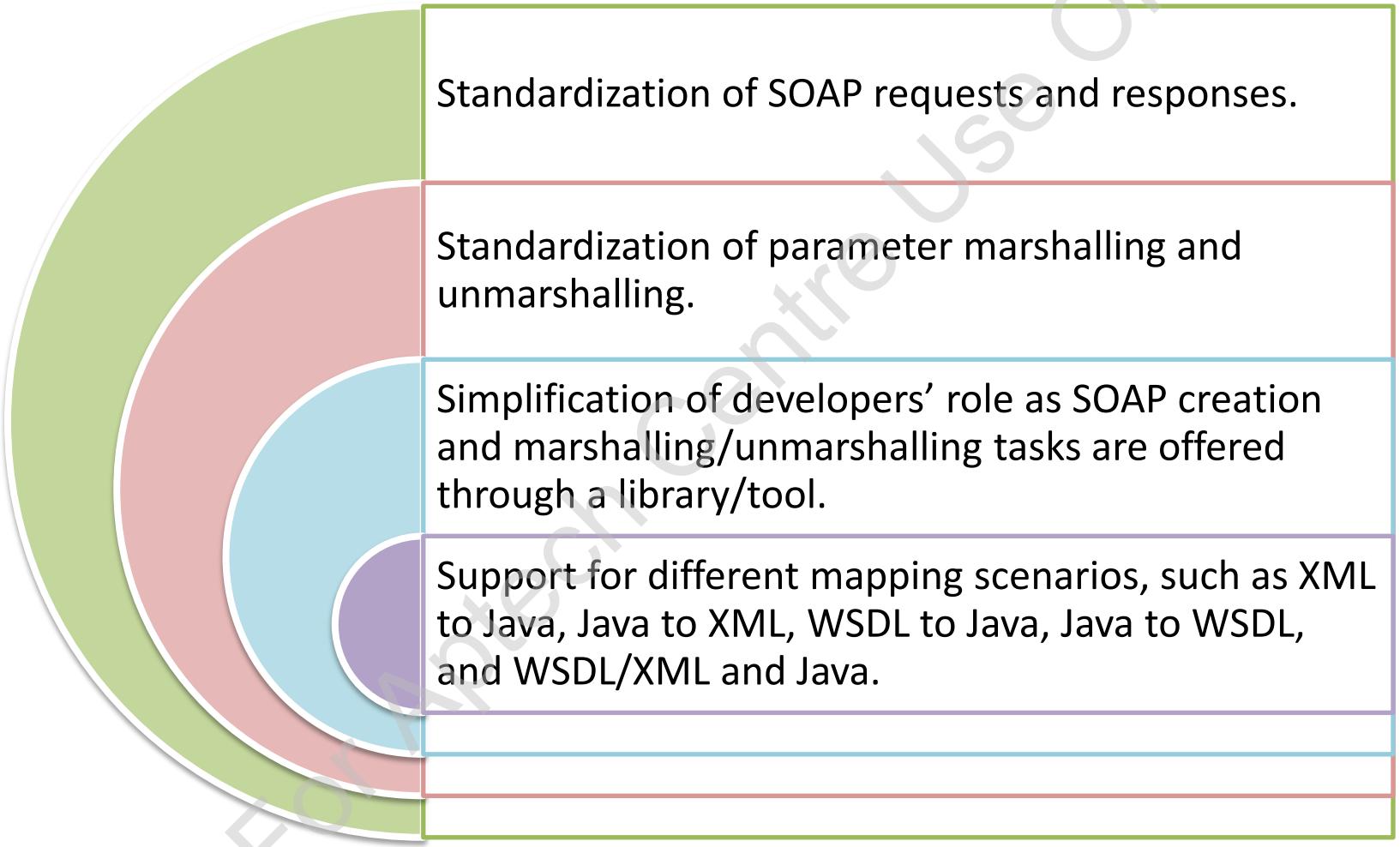
Provides portability of Service endpoints and service clients across JAX-WS implementations.

Defines a common programming model for Java EE endpoints and Web service clients.

Allows hosting of Web service endpoints that can be accessed by non-Java client applications.

Facilitates interoperation of Java EE applications as it is designed as a Java API.

Benefits of Using JAX-WS



Client Requests to JAX-WS Service 1-3

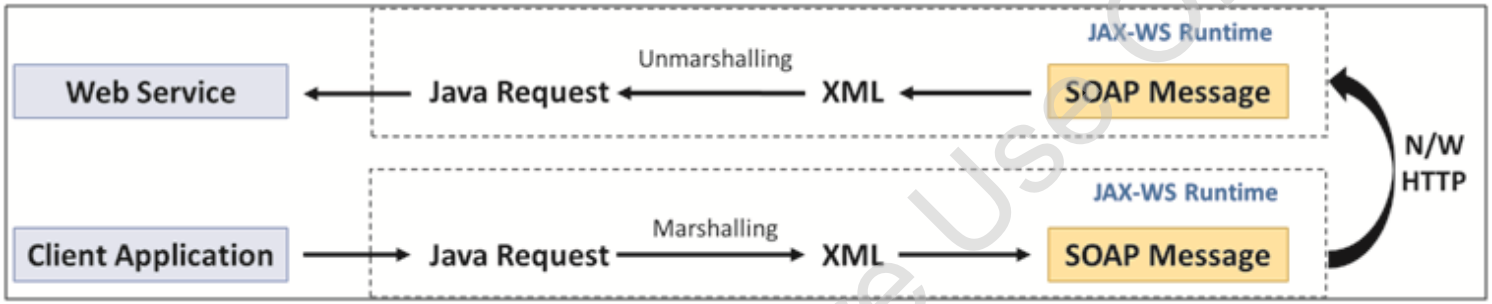
- Steps performed by the client application to use a Web service are as follows:

1. The request from client is passed through client-side JAX-WS runtime.
2. The runtime maps the request to XML and converts it to a SOAP form.
3. The message is then sent to a server on the network.
4. At the server, the SOAP message is received by a JAX-WS runtime.
5. The runtime applies the XML to Java mapping and maps the request with its parameters to the corresponding Java method.

- To make a request, the client application can use the pre-created static classes or classes/interfaces generated at runtime.

Client Requests to JAX-WS Service 2-3

- Following figure depicts the client requests to JAX-WS service:

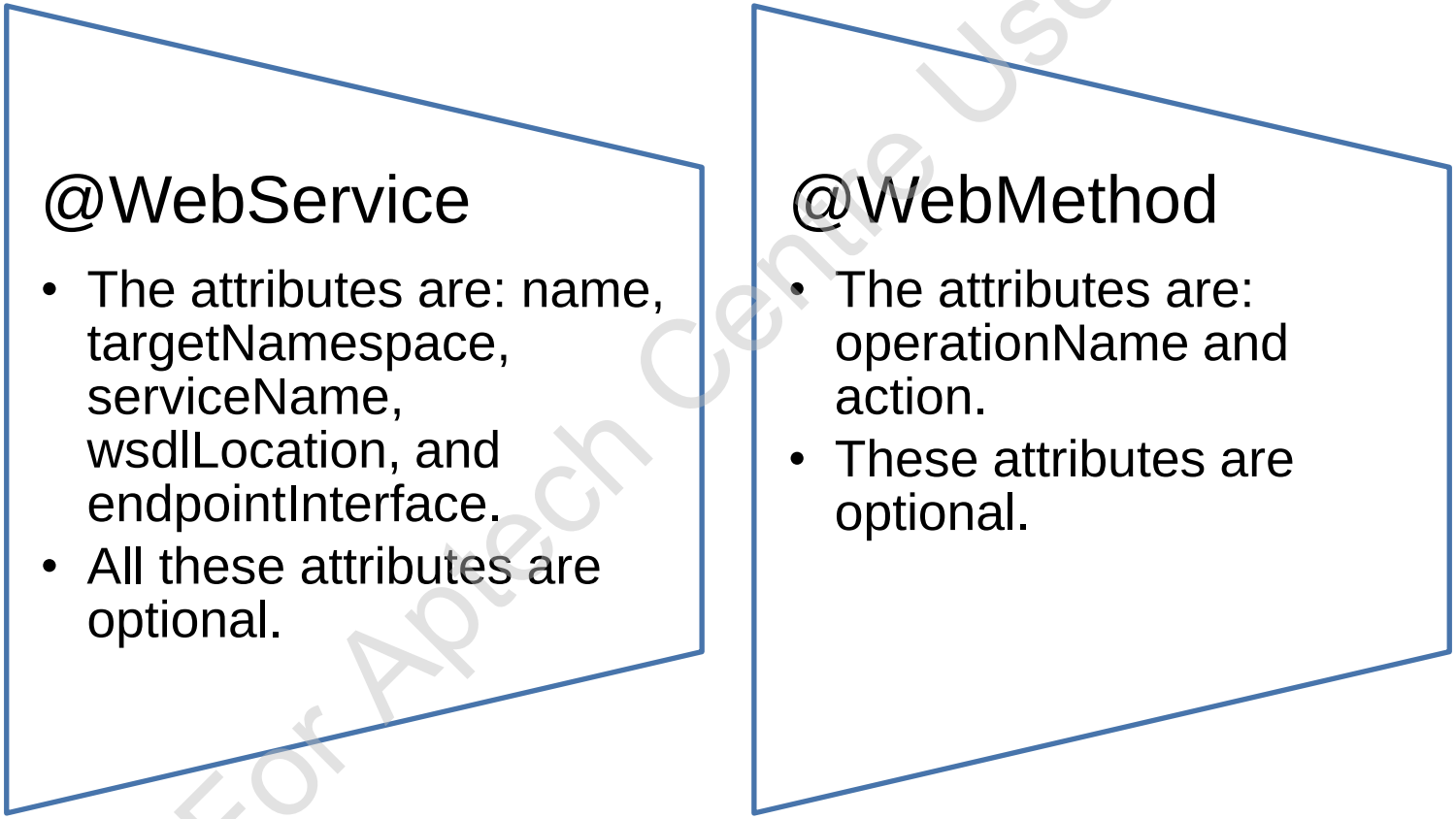


- Following Code Snippet shows how to implement JAX-WS:

```
import javax.jws.WebService;
import javax.jws.WebMethod;
@WebService
public class Welcome{
private final String message1 = "Welcome, ";
private final String message2 = "to WebService Course.";
public void Welcome(){
}
@WebMethod
public String sayWelcome(String name){
return message1 + name + message2;
}}
```

Client Requests to JAX-WS Service 3-3

- The code displays a welcome message to the users who call the Web service.
- Following two annotations have been used in the Code Snippet:



Registry Standards

A registry is a place where interfaces are published by the service to make the clients know about the service.

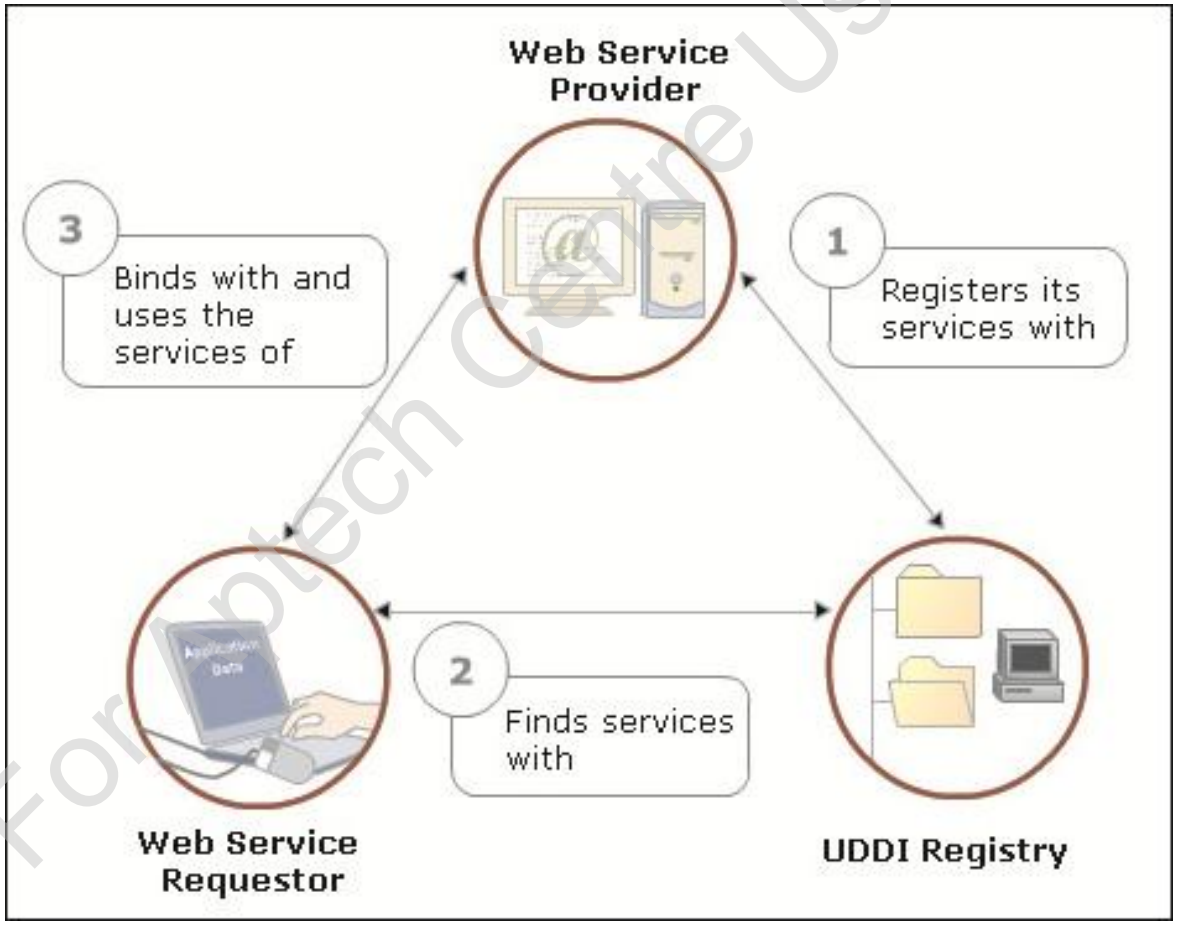
XML-based registries have standards for registering, deregistering, and looking up Web services across different platforms, systems, and languages.

The most popular registries are the Universal Description, Discovery, and Integration (UDDI) and the electronic business XML (ebXML) registries.

UDDI Registry Standards 1-2

UDDI has a standard mechanism to store information about organizations and their Web services.

- Following figure depicts the UDDI registry:



UDDI Registry Standards 2-2

- Following table describes the four core elements in the UDDI information model:

Element	Description
businessEntity	Describes all information including the name, description, and contact details of a business.
businessService	Groups related services of an organization.
bindingTemplate	Provides instructions to invoke a remote Web service.
tModel	Contains information such as name, publishing organization, and URL pointers to the actual specifications themselves.

ebXML Registry Standards

The Electronic Business XML (ebXML) is a business-to-business XML framework, which facilitates electronic business over the Internet.

ebXML registry

- Is a metadata registry and a repository that can hold arbitrary content.
- Has repository that can have metadata, technical specifications, and related artifacts.
- Stores information about Collaboration-Protocol Profile (CPP) and Collaboration-Protocol Agreement (CPA).

ebXML information model

- Enables data validation for improved integrity of registry data.
- Facilitates packaging (or grouping) of related registry objects.
- Allows both synchronous and asynchronous communication.
- Supports digital-signature-based authentication, validation, and authorization.

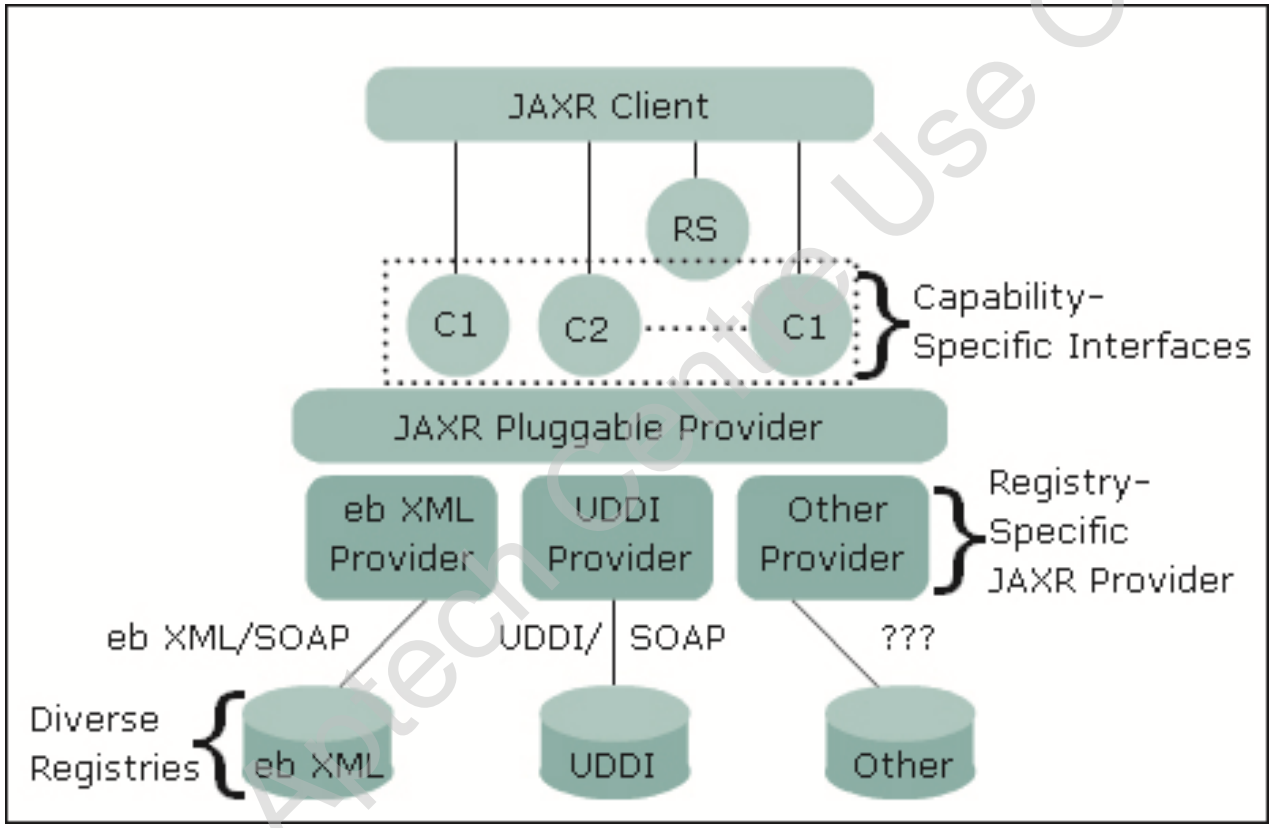
JAXR API

Java API for XML Registries (JAXR) API

- Is an abstract uniform Java API.
- Has a single set of APIs that can access many XML registries, including UDDI and ebXML registries.

JAXR Architecture

- Following figure shows the JAXR architecture:



- The JAXR information model is based on the ebXML Registry Information Model (RIM). It also supports UDDI and has all the data types defined in the UDDI Data Structure Specification.

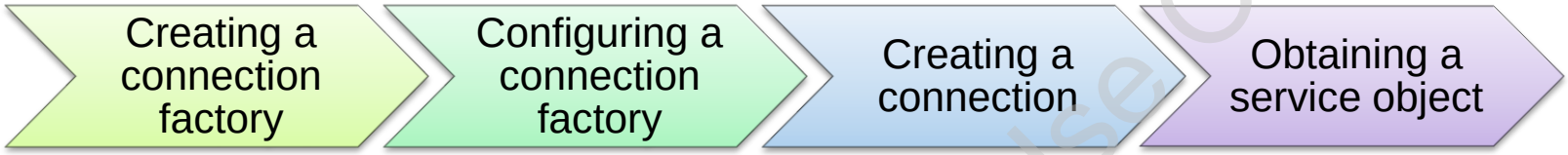
JAXR Information Model

- Following table lists some of the interfaces that define the JAXR information model:

Interface	Description
Organization	Organization instances are RegistryObjects that provide information on an organization that has been published to the underlying registry.
Service	Service instances are RegistryObjects that provide information on services offered by an organization.
ServiceBinding	ServiceBinding instances are RegistryObjects that represent technical information on how to access a specific interface offered by a Service instance.
Concept	Concept instances represent an arbitrary notion or concept. It can be virtually anything.
Classification Scheme	ClassificationScheme instances represent a taxonomy that can be used to classify or categorize RegistryObject instances.
ExternalLink	ExternalLink instances provide a link to content managed outside the registry using a URI.
ExternalIdentifier	ExternalIdentifier instances provide identification information to a RegistryObject.
Classification	Classification instances classify a RegistryObject instance using a classification scheme.
PostalAddress	PostalAddress instances provide address information for a user and an Organization.

Connecting to Registry

- A JAXR client establishes a connection with a registry by:



- After getting registry object, the service and service provider's information can be added by:

Publishing Data on Registry 1-5

- Following Code Snippet demonstrates the steps to publish a Web service to UDDI:

```
public static void main(String[] args) throws JAXRException{
try {
//Setting the properties for the ConnectionFactory Properties enviro = new
Properties(); enviro.setProperty("javax.xml.registry.queryManagerURL",
QUERY_URL); enviro.setProperty("javax.xml.registry.lifeCycleManagerURL"
, PUBLISH_URL);
enviro.setProperty("javax.xml.registry.factoryClass",
"com.sun.xml.registry.uddi.ConnectionFactoryImpl");
//creating a connection
ConnectionFactory confac = ConnectionFactory.newInstance();
confac.setProperties(enviro);
Connection conn = confac.createConnection();
//Authenticating the UDDI userName and Password
PasswordAuthentication passwdAuth = new
PasswordAuthentication(uddiUserName,
uddiPassword.toCharArray());
Set credentials = new HashSet();
credentials.add(passwdAuth);
conn.setCredentials(credentials);
```

Publish data to the UDDI by configuring the properties.

Then, create a connection.

Authenticate the UDDI credentials.

Publishing Data on Registry 2-5

```
//Obtaining a reference to the RegistryService,
//BusinessLifeCycleManager and the BusinessQueryManger
RegistryService registryservice = conn.getRegistryService();
BusinessLifeCycleManager lifecyclemgr =
registryservice.getBusinessLifeCycleManager();
BusinessQueryManager querymgr =
registryservice.getBusinessQueryManager();
//Creating an organization object
Organization company = lifecyclemgr.createOrganization("Aptech");
InternationalString description =
lifecyclemgr.createInternationalString("Leading IT
Trainers");
//Creating a user object
User contact = lifecyclemgr.createUser();
PersonName name = lifecyclemgr.createPersonName("John Miller");
contact.setPersonName(name);
//creating and assigning the users telephone number
TelephoneNumber telnum = lifecyclemgr.createTelephoneNumber();
telnum.setNumber("1800-420-8000");
Collection phonenumbers = new ArrayList();
phonenumbers.add(telnum);
contact.setTelephoneNumbers(phonenumbers);
```

Obtain a reference to the RegistryService, BusinessLifeCycleManager and the BusinessQueryManager classes.

Publishing Data on Registry 3-5

```
//creating and assigning the users email address
EmailAddress email = lifecyclemgr.createEmailAddress("enquiry@abcinc.com");
Collection emaillist = new ArrayList();
emaillist.add(email);
contact.setEmailAddresses(emaillist);
//Setting the user as the primary contact for the organization
company.setPrimaryContact(contact);
//Creating the Web service object
Collection servicelist = new ArrayList();
Service service = lifecyclemgr.createService("Courses");
InternationalString serviceDescription =
lifecyclemgr.createInternationalString("Available
courses..");
service.setDescription(serviceDescription);
//Creating the Web service bindings
Collection serviceBindigs = new ArrayList();
ServiceBinding binding =
lifecyclemgr.createServiceBinding();
InternationalString bindingDescription =
lifecyclemgr.createInternationalString("Course Name");
```

Define the data that needs to be published to the UDDI.

Define the Web services to be offered and the Web service bindings for the same.

Publishing Data on Registry 4-5

```
binding.setDescription(bindingDescription);
binding.setAccessURI("http://www.aptech.org");
boolean b = serviceBindigs.add(binding);
service.addServiceBindings(serviceBindings);
servicelist.add(service);
company.addServices(servicelist);
//Classifying the organization
ClassificationScheme scheme = querymgr.
findClassificationSchemeByName(servicelist,"Courses");
Classification classification = lifecyclemgr.
createClassification(scheme, "Course name", "Course number");
Collection classificationlist = new ArrayList();
boolean c = classificationlist.add(classification);
company.addClassifications(classificationlist);
Collection organizationlist = new ArrayList();
organizationlist.add(company);
//making the final call to the registry to get a response
BulkResponse response = lifecyclemgr.saveOrganizations(organizationlist);
Collection exceptions = response.getExceptions();
```

Classify the Web services offered into the appropriate category.

Publishing Data on Registry 5-5

```
if (exceptions == null) {
Collection keys = response.getCollection();
Iterator iterator = keys.iterator();
Key key = (Key) iterator.next();
String uid = key.getId();
company.setKey(key);
}
else { // there is an exception
Iterator iterator = exceptions.iterator();
while (iterator.hasNext()) {
Exception exception = (Exception) iterator.next();
System.out.println("Exception...." + exception);
}
}
conn.close();
}
catch(Exception e) {
e.printStackTrace();
}
}
```

Publish all the data into the registry and awaits response from the registry.

Registry throws exceptions in case any errors are encountered.

If there are any exceptions, the details of the exceptions is displayed in the console.

Else, the data is published and added to the Web service.

Querying the Registry

- A registry can be queried by using organization and service names related to the organization that needs information.

Finding Organizations by Name

- A find qualifier collection is created to store find qualifiers.
- A constant is added to show that the search results have to be sorted in descending order.
- A name pattern collection namePat is created and the string of organization name is added to it.
- Using find qualifier and name pattern instances, the findOrganizations() method is passed.

Finding Services and Service Bindings

- An iterator instance is created from the collection returned by getCollection() method.
- A while loop is defined to process all the organization objects in the iterator. Inside the loop, the organization object is retrieved using the next() method.
- An iterator is created from the collection returned by getServices() method.
- While loop is defined to process each service associated with the object.

- One can remove any data submitted by using the key returned by the registry as an argument to one of the BusinessLifeCycleManager delete methods.
- The different delete methods include deleteOrganizations(), deleteServices(), deleteServiceBindings(), and deleteConcepts().

SAAJ 1-2

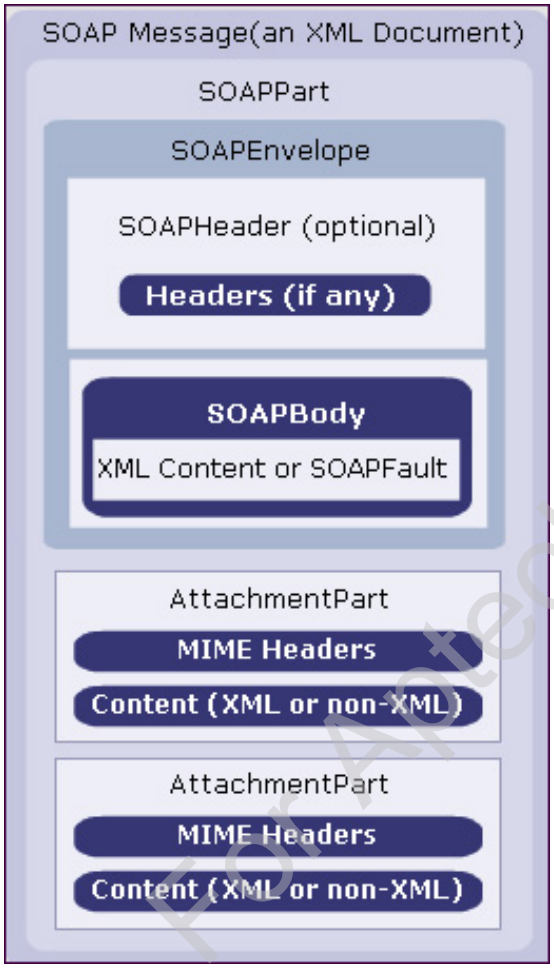
SOAP with Attachments API for Java (SAAJ) is an API that allows users to create and send SOAP messages with attachments by means of the javax.xml.soap package.

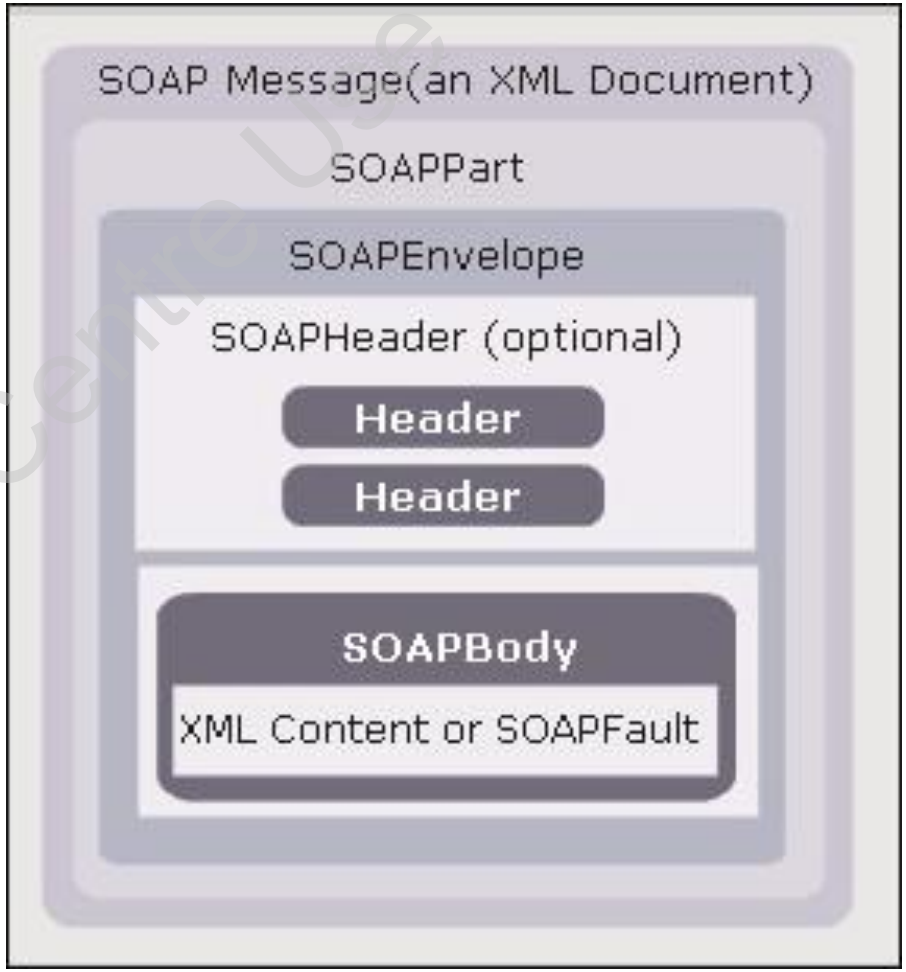
SOAP provides a common message format for Web services. It enables developers to produce and consume messages conforming to the SOAP 1.2 specification and SOAP with Attachments note.

- Using SAAJ API, one can create XML messages that adapt to the SOAP 1.2 and WebService-I Basic Profile 2.0 specifications.
- There are two perspectives of SAAJ, namely Messages and Connections.

SAAJ 2-2

- There are two types of SOAP messages are those that have attachments and those without attachments, which are as follows:





SAAJ API

- The package javax.xml.soap provides the API for creating and building SOAP messages.
- Following table describes some of the classes that constitute SAAJ API:

Class	Description
AttachmentPart	A single attachment to a SOAPMessage object.
MimeHeader	An object that stores a MIME header name and its value.
SOAPConnection	A point-to-point connection that a client can use for sending messages directly to a remote party.
SOAPConnectionFactory	A factory for creating SOAPConnection objects.
SOAPMessage	The root class for all SOAP messages.

SOAP Connection

- In SAAJ API, the connection is represented by a SOAPConnection object, which goes from the one endpoint to another endpoint.
- Following Code Snippet creates the SOAPConnection object con and then uses it to send the message:

```
SOAPConnectionFactory fact = SOAPConnectionFactory.newInstance();
SOAPConnection con = fact.createConnection();

// create a request message and give it content

java.net.URL endpoint = new URL("http://java.sun.com/javaee/7/docs");
SOAPMessage response = con.call(request, endpoint);
```

- All the messages sent over a SOAPConnection object are sent with the call() method.
- The return value for the call() method is the SOAPMessage object that is the response to the message that was sent.

JAXB

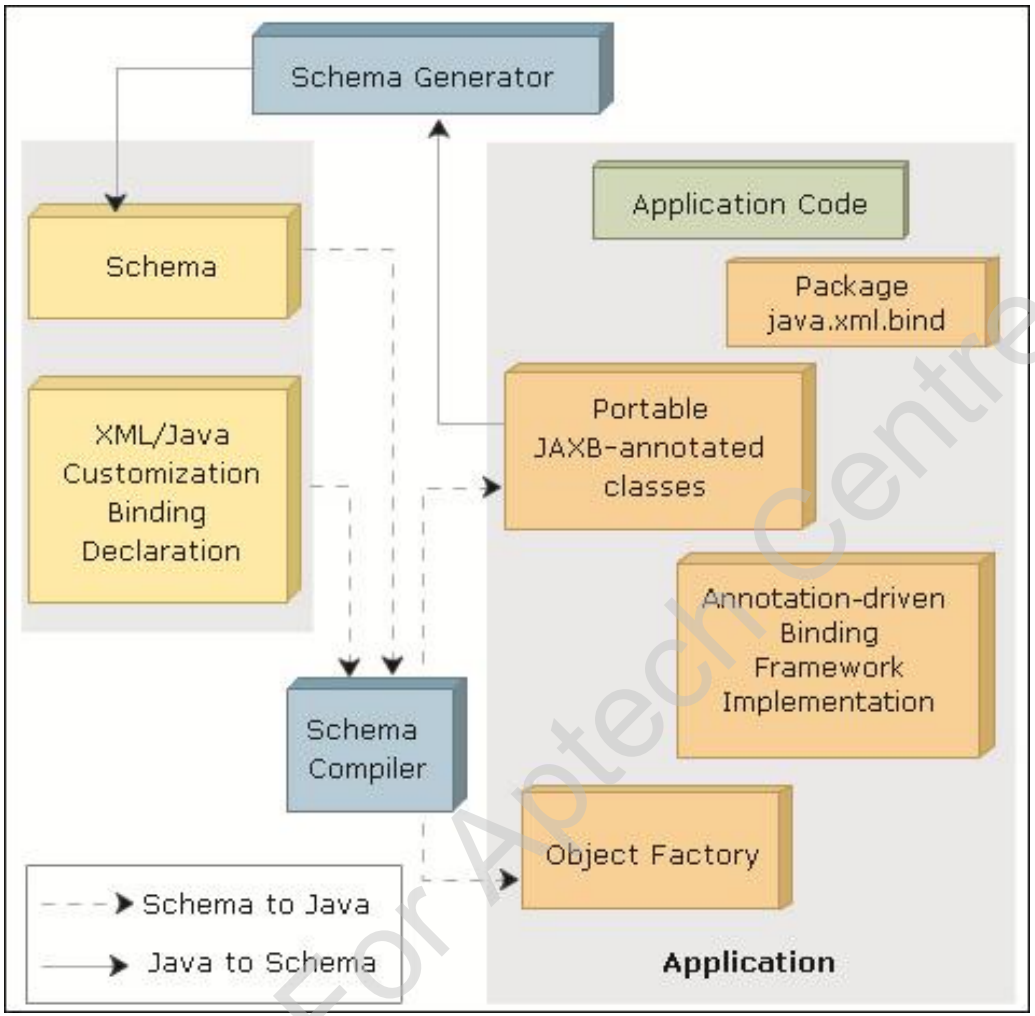
Java Architecture for XML Binding (JAXB) is a set of interfaces which helps client applications to communicate with code generated from a schema.

- Following table shows the three components of JAXB:

Interface	Description
Binding compiler	Creates Java classes from a given schema.
Binding framework	Provides runtime services such as marshalling, unmarshalling, and validation that have to be performed on the contents classes.
Binding language	Describes schema binding of Java classes using which you can override the default binding rules.

JAXB Architecture

- Following figure shows the JAXB architecture:

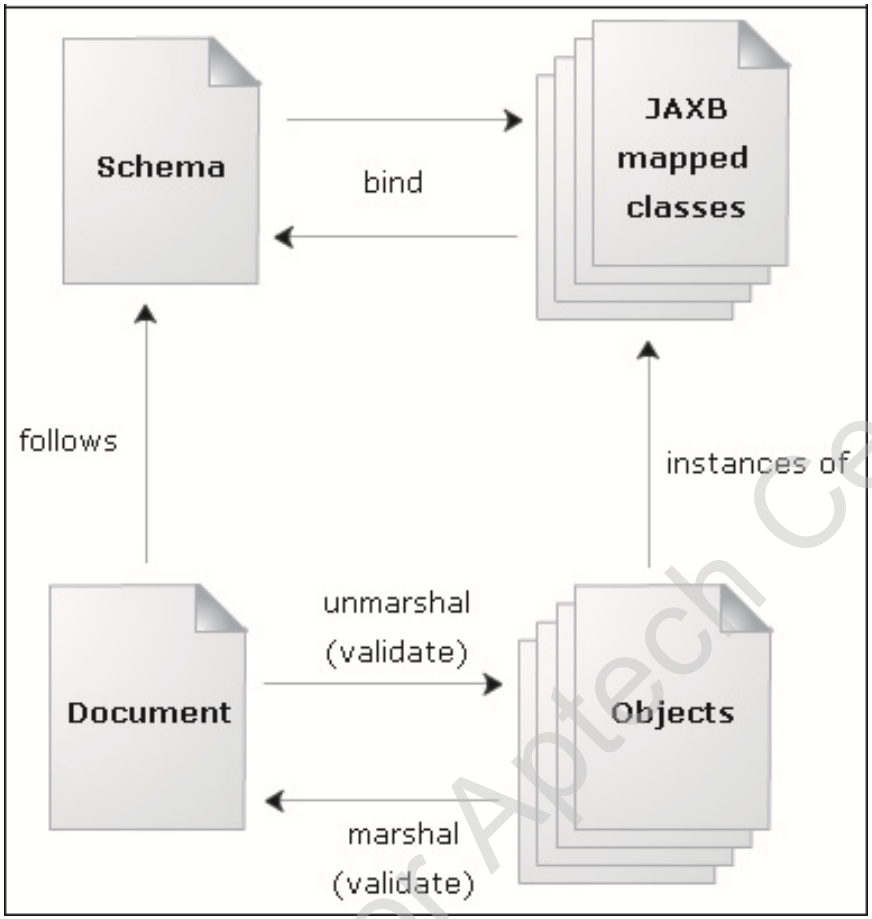


The architectural components of JAXB are as follows:

- Schema compiler
- Schema generator
- Binding runtime framework

JAXB Binding Process

- Following figure shows the JAXB binding process:

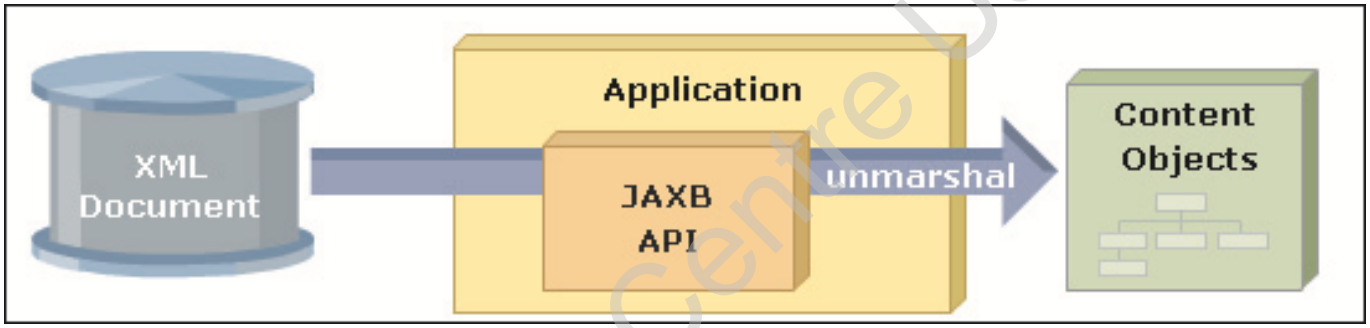


Steps in the JAXB data binding process are as follows:

1. Generate classes
2. Compile classes
3. Unmarshal
4. Generate content tree
5. Validate
6. Process content
7. Marshal

Concept of Unmarshalling

- The process of converting an XML document into a content tree is termed as unmarshalling.
- Following figure depicts unmarshalling:



- JAXB allows to access XML data without unmarshalling it.

Unmarshalling

- Following Code Snippet shows how to parse an XML document using a DOM-based parser:

```
//Step1
import javax.xml.bind.JAXBContext; JAXBContext jc =
JAXBContext.newInstance(
"Information.jaxb");
//Step2
import javax.xml.bind.Unmarshaller;
Unmarshaller unmarshaller = jc.createUnmarshaller();
//Step3
Collection collection = (collection)unmarshaller.unmarshal(new
File("products.xml"));
//Step4
CollectionType.ProductsType productsType = collection.
getProducts();
List productList = productsType.getProduct();
```

1. An object of JAXBContext class is created whose context path is Information.jaxb.
2. An object of the Unmarshaller class is created that controls the process of unmarshalling.

3. The unmarshal() method is invoked, which performs the actual unmarshalling of the XML document, products.xml.
4. The get() method in the schema-derived classes is used to access the XML data.

Concept of Marshalling

- In the marshalling process, the user has to build an XML document using a content tree.
- Following Code Snippet demonstrates how to create an XML document from a DOM tree:

```
//Step1
import javax.xml.bind.JAXBContext;
JAXBContext jc = JAXBContext.newInstance(
                 "Information.jaxb");
//Step2
import javax.xml.bind.Marshaller;
Marshaller marsh = jc.createMarshaller();

//Step3
marsh.setProperty(Marshaller.JAXB_FORMATED_OUTPUT, new Boolean(true));

//Step4
marsh.marshal(collection, new Fileoutputstream("Output.xml"));
```

1. An object of the JAXBContext class is created and the appropriate context path of the package that contains the classes and interfaces for the bound schema is specified.
2. An object of the Marshaller class is created which controls the process of marshalling.

3. The Marshaller object sets its properties using the setProperty method.
4. The marshal() method is invoked by specifying an object that contains the root of the content tree and the output target.

Data Validation

- Validation is the process of verifying if an XML document meets all the constraints expressed in the schema.
- It has to be done stringently while writing out data, but not when reading data.
- During unmarshalling, setValidating() method has to be used to validate source data against the associated schema.
- If an error is encountered, the data processing need not be stopped.
- Instead, a validation error report should be generated.
- JAXB specification authorizes all providers to report validation errors.
- If an XML document is found invalid, it will still be unmarshalled although the result will not be valid.
- Validation is not done as part of marshalling process as setValidating() method does not exist for marshalling.
- Validation can be done at one time and marshalling at another time.

Limitations of JAXB

Limitations of JAXB:

- JAXB cannot be used to process generic XML as it requires a DTD and a subset of XML Schemas.
- Additional work is required to tell JAXB what kind of tree it should construct to simplify the application.
- JAXB does not support the legal DTD constructs.

Summary

- Java EE Web services are platform-independent and language-independent services available on the Web.
- Service Oriented Architecture (SOA) aids application development by a loosely coupled modular approach. It permits the applications to extend and utilize services.
- SOAP is a Web service standard for packaging the XML data that are transferred between applications.
- The Java EE platform has two types of endpoints for Web services. One extends Stateless Session Beans into Web services and the other resembles a plain Java class.
- The Java API (JAXP) provides a framework for using SAX2 and DOM2 that read, write, and modify XML documents.
- JAX-WS provides a framework and runtime environment to create and execute XML-based Web services.
- The Java API for XML Registries (JAXR) API provides a single set of APIs to access a variety of XML registries.
- The Java Architecture for XML Binding (JAXB) API is a set of interfaces that enable communication through the code generated from a schema.