realspeed

Web Services

1.What is Web Service?

➔Web Service is something that is related to Distributed Technology.

2.What is Distributed Technology?

➔In distributed computing, processing and data are spread out over multiple computers, usually over a network.

3.When we should go for distributed technology?

- **When you want HIGHER THROUGH-PUT out of your system then go for Distributed Technology.**

Example: Within 1hr of time we are able to complete 10 jobs. Now we are want 20 jobs to complete within same amount (1hr) of time i.e. Called Through-Put.

If we want to increase the Through-Put of our application we must go for Distributed Technology.We can add some extra G.B of RAM to our Systems for achieve it.

- **If we want to reduce the TURN AROUND time for a job then we should go for Distributed Technology.**

Example: If a job takes 100 hours of time to complete, if we distribute this job across multiple machines and executed parallely, it would be completed much faster than 100 hours (50hrs).

- **To make your systems to be highly responsive then you have to go for Distributed-Technology.**

Example: Train ticket booking before the train leaving from the railways platform.

- **If you want HIGHLY AVAILABILITY OF RESOURCES then we should go for Distributed-Technology.**

Example: If you assign a job to 1 single person he may not able to handle the job. IF there is no other person who will continue to complete your job then your job will be halted or paused.

Instead of it if you assign your job to multiple resources i.e. one can share the load of the other people & can complete the job

- **If you want the HIGHLY UTILIZATION OF RESOURCES then we should need to go for Distributed-Technology.**

➔If in a office 1 single computer access a printer it may not utilize printer maximum , So that if we distributed that printer to multiple other computer then you can maximum utilize the particular printer.

4.Why Java has provided support for Distributed Technology?

➔Every programming languages added support to build distributed applications to make them more superior than other languages.

➔Now programmer does need to worry about how to make my applications communicated over network. Use that java provided APIs so easily develops the application to make your application communicate over the network

5.What are the APIs that are been provided by SUN MICRO SYSTEM as part of the Java language?

1. **CORBA (Common Object Request Broker architecture)**
2. **RMI (Remote Method Invocation)**
3. **EJB (Enterprise Java Bean)**
4. **WEB SERVICES**

6.CORBA (Common Object Request Broker architecture)

➔ **CORBA stands for Common Object Request Broker architecture**.
➔CORBA meant for exposing any language object/application to be a Distributed Program/Application**.**
➔ CORBA **encourages the developers to writing their classes** or representation of applications in a language **neutral format**. So that CORBA encourages every language object to make as Distributed.
➔ **In CORBA programmer has to code using IDL (Interface Definition Language).**

➔ **IDL describes the skeleton representation of your classes and their methods and their parameters and their return type in the IDL scripting file**.

➔So that we don't need to write the logic for programming to make your application for communicate over the network**. We just have to tell to the IDL file which piece of business logic** & **which set of inputs from where**.

7.How To Work with Corba

➔It is a scripting language where developer **writes the IDL file and gives it to CORBA compiler.**
➔The **CORBA compiler will generate the language specific object**.
➔The programmer has to write the business logic. After writing the business logic, compile your classes again with java compiler and get the byte code of those classes then in order to expose the CORBA object over the network and to **make your classes always available or alive for others it has to be deployed on MOM's**.

➔ **MOM(Message Oriented Middleware)** is a CORBA server.

➔The purpose using of MOM is to host CORBA objects. But in order to use MOM, we need license means it is not open source.

8.Disadvantages of CORBA

Considering all the above factors.
➔Like development will starts with IDL (seems to be different from general way of programming)
➔Deploying requires a licensing server, at those times working with CORBA seems to be complicated.

This makes CORBA quickly vanish from the market**.**

9.RMI (Remote Method Invocation).

➔ **RMI only supports Java language specific objects** only to **expose as Remote Objects over the network across the world.**

➔ **Anyone can call the functionality which we wrote inside the method** on that object across the world.

➔ Now those functionalities or business logic which you written inside the method as part of the class can **not only used by other classes running on the same JVM** or same machine ,rather it **can accessed by any other classes running on any other JVM** or **any other machine over the network across the world**.

➔ In RMI java developer write the business logic in the POJO class, compile to **RMI Compiler, which will generate class file to expose over the network.**

10. Advantages of RMI

➔ Here the programmer need to worry to code a different language file or need not write the code in some other generated object, rather the development starts with POJO and once finishes will generate network abstractions to expose it.
➔ To expose the generated object over network, it has to be deployed on RMI server (In memory server/container).
➔The RMI server is open source and would be shipped as part of JDK and is very light weight server.

11.EJB (Enterprise Java Bean)

➔EJB stands for Enterprise Java Bean.
➔ It is superior than RMI but it is not the replacement of RMI. It is alternate of RMI.
➔So EJB is a heavy weight component where RMI is a light weight component.

12. Why EJB is Heavy Weight component.

➔EJB container is provided by vendor.
➔ EJB Technology provides a container called EJB container.
➔ All the normal standard java class packaging model (simple jar/ear application) will allow.
➔ Whenever a client has sent a request, EJB container is the responsible for receiving and handover that request.
➔If there are multiple EJB are available inside 1 container like servlet. Then we have to configure those in deployment descriptor (EJB-jar.xml).
➔While your EJB object is executing, it might need some external infrastructural resource support, instead of programmer coding for it, (like security or transaction or auditing or connection pooling), these would be configured, maintained and provided to the EJB object by EJB Container.

➔So EJB is called as Heavy weight application

13.How many Type of EJB are there ?

➔ There are 3 types of EJB are there.
- ❖ **Session Bean**
- ❖ **Entity Bean**
- ❖ **Message Driven Bean**

Session Bean:

➔EJB session bean will make your Business functionality will be accessible over the network across the world.

Entity Bean :

➔Whenever you need the data from the database rather than you going to database using JDBC API , you will use the Entity bean. Entity bean will query the data and returns the data in the form of objects to us.

Message Driven Bean:

➔It is used for enabling 2 applications to communicate each other in disconnected way.

➔Here two applications communicate with each other in store and forward mechanism (i.e. the information which is stored in the messaging box will be forwarded to the targeted application who has to receive the messages.)

14. Limitation of EJB:

➔**If u expose the EJB over the network even though it is visible to everyone around the network but will not be accessible by everyone. Because if the other person who has to access the EJB should he must have the same technology (JAVA ONLY).**

To overcome this problem Web Services came.

Web Services

What is Web Service?

➔Web Service is something that is related to Distributed Technology.
➔ Web services allow a program to expose objects over the network.
➔ Web Service Distributed object are not only platform independent, and language independent.
➔We can build distributed interoperable Programs using Web services.
➔Anything that is accessible irrespective of Platform and programming language is Called Interoperability.
➔Which means you can write a Web service Object using C, C++, Perl, Phython, PHP, Java or .Net and can expose over the network. And can be accessed by any programming languages.

Where to use Distributed Technology?

Example 1:

➔In banking sector (HDFC credit cards).
➔Suppose my program or system has to generate credit card statements for all the costumers of HDFC bank (10 millions of customers)
➔Within a limited period of time (within 8hrs) this program will not generate the credit card statements on a daily basis.
➔It has to generate the statements on a particular time or day of a month.
➔Suppose it has to generate the statement on every 6th of a month.
➔It is a time bounded job.
➔So if there is a slippage of time then time taken to generate the bill will not be sufficient
➔And it took 1 more day extra then interest free day will be extended.
➔1more day will be added to the interest free day which is huge amount of revenue loss to the bank.

➔So this job has to complete in a specific restrict amount of time.
➔Within the 8hrs of time I wanted to create 10 millions of credit card statements for the card holder.
➔If I assign this job to a single person then he will not able to complete this job within the specified time.
➔I want the job to be completed but I want the time to be reduce in complete the job (Turn-Around time).
➔For this we have to go for Distributed Technology. We have to share this job across multiple people or system.

Example 2:

Let's talk about how Indian Railways (IRCTC) & its business partners (MakeMyTrip, Paytm, Yatraa.com) are work together?

➔Not only by using IRCTC we can book the ticket by using MakeMyTrip & Paytm also.

➔ Paytm & MakeMyTrip are the business partner of the IRCTC (owner) so they must have to follow the set of rules and guidelines which are provided by IRCTC for doing the business in association with IRCTC.

➔ MakeMyTrip & Paytm needs the underlying database or other internal information for doing the business with IRCTC so IRCTC has to expose database or other internal information to the MakeMyTrip & Paytm.

➔ If IRCTC is going to expose his database then there will be several problem will generate.

- ⇨ Security bridge
- ⇨ Complexity to integrating those system or application.
- ⇨ Maintainability issues.
- ⇨ IRCTC is defines complex business process of the business system
- ⇨ Those have to understand & adopted by the business partners which is not so easy.
- ⇨ Implementing such complex business system is not easy , It is more time consuming and huge amount of investment has to be done for the little returns.
- ⇨ So IRCTC has to provide the such kind of business information to the business partners, but it is almost same as hand overing the whole business to the business partners.
- ⇨ It is not so easy to implements all the rules & conditions across all the business partners so that the IRCTC has to evaluate & verify
- ⇨ If there is a small change within the business rules then that has to implemented across all the business partners.

➔So Without exposing the underlying database how will the MakeMyTrip & Paytm would be able to reserve the ticket on behalf of the customer who are trying to reserve the ticket with Paytm & MakeMyTrip?

➔ Instead of exposing the database or instead of providing the information about the business to the other people because of security problem.

➔ If I can somehow some making expose the components in which already they contains the business logic to the business partners then I don't need to give the database information to the external business partners or I don't need to tell the internal information about my business to the other people. Then integration efforts will become easy.

➔So Distributed technology comes into picture.

How can these components are in IRCTC can be used by MakeMyTrip or Paytm?

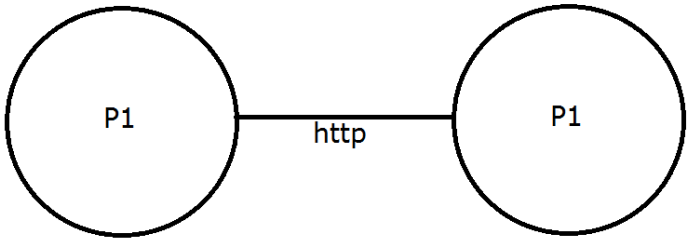
➔ If we built this components in CORBA, RMI or EJB then MakeMyTrip & Paytm are also has to be developed in java. That's the biggest limitation for these technologies.

➔ **Instead of this if we built these application components over using Web-Services**

➔Then irrespective of performance irrespective of platform the MakeMyTrip & Paytm could be running on any other platform and built on any other language still they should be able to access the Web-Services that is been exposed by IRCTC.

➔ Then integrating will become very easy.

How do Web-Services works internally?



➔ To exchange their information from one person to other, they need medium.
➔ Medium acts as a physical channel on which data would be exchanged.

Protocol:

➔If two computers want to exchange data, first they should be connected over a wire or network. This wire / network works as a medium.
➔ Even though there is 1 wire or 1 network connection it just makes the 1 wire acts as 10 wires, by dividing the physical communication channel into in to logical connection. So protocol is mandatory.
➔ Protocol guaranties the delivering of information or physical bits of information that is been send by 1 system will be guaranty to be deliver to the other system in an understandable format. So this the purpose of protocol.
➔ So protocol is a set of rules that protects those to systems who wants to exchange the information over the network.

➔ If two computer systems want to exchange data, having network itself is not enough we need a protocol to guard the exchange of data over them.
➔ Protocol simulates that everyone has their independent bands of communication channel to exchange information, even though everyone is sharing the same physical channel.
➔ There is specific protocol over which web service is going to work. **Web-Services only support the_Http (Hyper Text Transfer Protocol) as the protocol**. The initial version of Http is 1.0 & current version is 1.1.
➔ Web-Service also supports several other protocols. Like FTP, SMTP, UTP, JMS. But there is some term and conditions to use this .Because these are not interoperable.
➔ Protocols are firewall enabled, which makes our Web Service application more secure

Example:1

➔Let's say two people are talking to each other, instead of one person listening to other both the people started talking to each other at the same time.
➔It seems instead of talking there are barking at each other.so that communication will be effected.
➔ Rather if there are set of rules that guards their communication,
for e.g.. First person says am starting the conversation so that the other person would be in listening mode.
➔ once the first person finishes, he will say am done so, that the other person can start the conversation by following the same rules.

➔ This indicates we need set of rules that guards the communication channel which is nothing but a protocol.

Firewall

➔Securing our computer/application that not to allowing every program to open the port

Character Set Encoding

➔ To exchange the data between two machines in an understandable manner which is ASCII Code or UNICODE encoded characters which are common language format for all.

➔If a system that is sending the data using ASCII or UTF-8 as character set encoding, then the other system who is receiving the data has to follow/use the same character set encoding.

XML

➔ We can represent the data in MS-Word, MS-Excel etc. But the problem with these kinds of representations is those are recognized by only windows and few other platforms, which means those are platform dependent formats

➔But we are trying to build an interoperable solution, which means the data we carry between systems must also be interoperable.

➔ There is only one way of carrying data between systems in an interoperable manner is XML.

➔ **Interoperable means irrespective of language & irrespective of platform it can used anywhere.**

➔XML stands for **EXTENSION MARKUP LANGUAGE**. It is built over tags.

➔ The first and current version of XML is 1.0.

➔ XML is not a programming language.

➔It is used for storing the data. It stores the data in its own XML format.

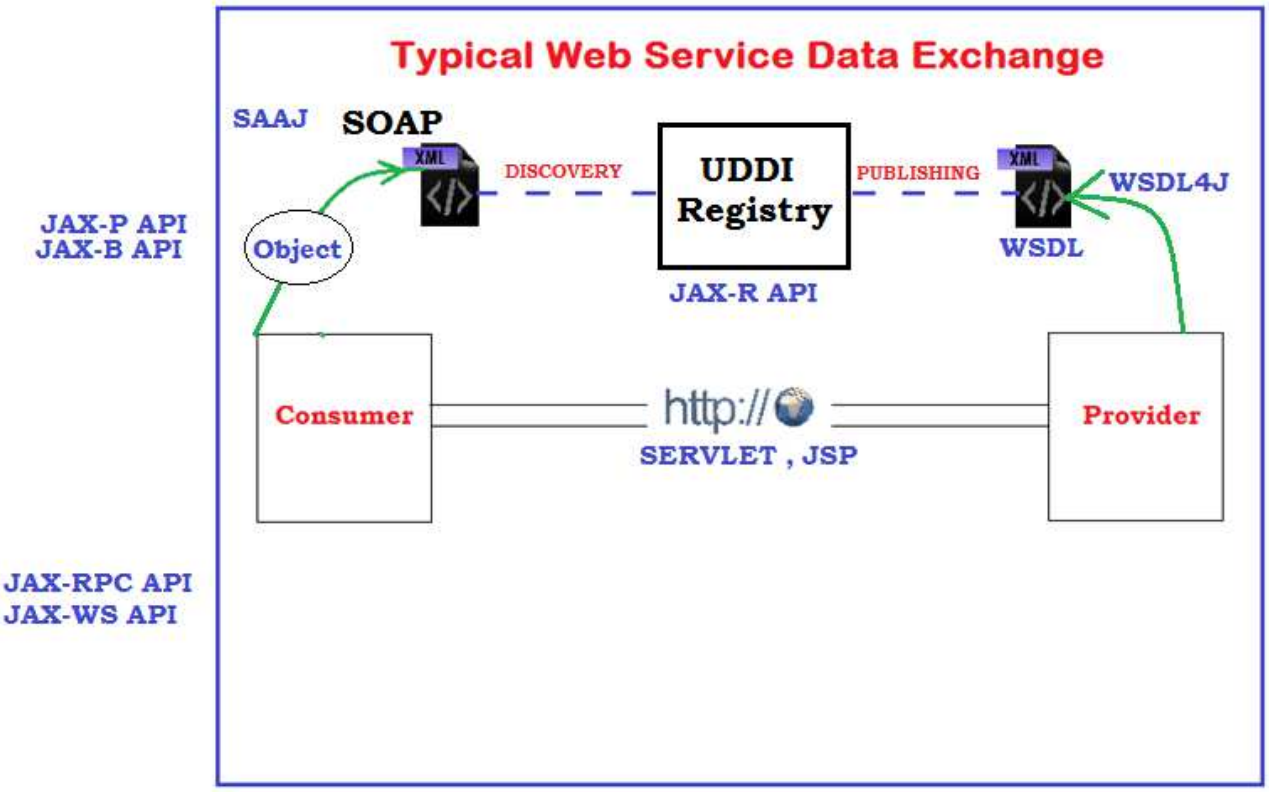
➔ XML stores the data in a structured format. It also semantics attached to the data as well. So anyone can easily identify the data.

➔ In the above XML it is clear that which data is conveying what type of information, So that the receiver of this will not have a chance of miss-interpreting the information. So, the default standard for exchanging information between computers is XML.

Ex:

```
1  <employee>
2      <empId>111</empId>
3      <empName>AAA</empName>
4      <empSal>3000</empSal>
5      <empAddr>Hyd</empAddr>
6  </employee>
7
```

Web-Service Architecture:



WSDL (Web Service Description Language):

➔WSDL Stands for Web Service Description Language. It is also a special XML type document, this makes it interoperable.
➔ To access the functionality of the provider, consumer should know the information about provider.
➔ That means not only the program the documentation should also be Interoperable.

➔ WSDL has a pre-defined structure and has grammar, pre defined elements, fixed order for writing those elements, tags using which you need to write the WSDL document.

➔ WSDL document will provides the entire information about the provider like Service name ,Class Name, Number of Methods, their parameters and return types and Location of the Class ,which method gives which functionality etc. The consumer can know the entire information about the provider by seeing its WSDL document.

UDDI (Universal Description and Discovery Integration Registry):

➔ UDDI stands for "Universal Description and Discovery Integration Registry", this is the registry which store's all the WSDL documents in it.

➔UDDI should be accessible irrespective of programming language which means These should also be interoperable, that's why those are built using XML technology, so those are also called as XML registries.

➔ Provider will publishes the WSDL documents to the UDDI registry and Consumers will search for a WSDL document from the registry.

Evolution of Web-Services:

➔To build interoperable distributed applications, every software vendor in the industry started defining their own standards.

➔SUN has defined its own standards, Microsoft defined its own standards and IBM came with its own standards etc.

Vendor	SUN	Microsoft	IBM
Transport Protocol	HTTP	FTP	SMTP
Language for exchanging data	XML	MSXML	CSV
Description	JWSDL	MSWSDL	IBM4SDL

➔ In this case Interoperability never will become a reality. Finally it was realized by Every software vendor that we cannot build independently an interoperable distributed applications, unless everyone has agreed upon same standards.

➔So They comes an organization **WS-I**, stands for **Web Service Interoperability org anization.**
➔ It is a non-profitable organization which is formed to build open standards for building interoperable distributed applications. The members of this group are people/representatives from various vendors in the software industry.

➔ **WS-I** has released **BP 1.0** (Basic Profile) specification document. The specification document comprises of set of guidelines or standards that must be used to build an Interoperable distributed programs.
➔ After few years of BP 1.0, WS-I has again released one more specification document BP 1.1 which is the successor of BP 1.0 which contains some advancements than its earlier.

Java API's for WS-I Specification:

➔ **WS-I's BP 1.0** and **BP 1.1** specification documents contains guidelines for building Web Services in any programming language, those are not specific for Java. SUN has to adopt those guidelines to facilitate building of Web services in Java.

➔To build **WS-I's BP 1.0** specification complaint Web Services SUN has released **JAX-RPC API.**
➔ To build **BP 1.1** specification complaint Web Services **JAX-WS API** has been released.

➔**BP 1.1** is the successor than **BP 1.0**, similarly **JAX-WS API** is the successor than **JAX-RPC API**.

How Web Service works

➔Consumer is the person who tries to access the information from the Provider, and Provider is the person who always services the Consumer.
➔ Consumer and Provider want to exchange information, so first they should be **connected with a wire or over a network** and **needs a protocol to guard their communication**. **The recommended protocol in a web service communication is HTTP. HTTP protocol is firewall friendly**.
➔They need a common language to exchange data, **the default standard for exchanging information between computers is XML**
➔But only XML is not enough, as a **Web Service communication not only transmits business data**, it might also transmit helper data(like OTP) or some extra processing information, so **we need a layer on top of it called SOAP.**
➔SOAP stands for "**Simple Object Access Protocol**" acts as a binding protocol to classify the information.
➔ If the consumer wants to access the provider, he needs to know the information About the provider
➔ **So, the information about the provider is documented in a document called WSDL and is being published by the Provider in a registry.**
➔ **So the registry in which WSDL document is stored by the provider is called UDDI Registry.**
➔ **The process of putting the document into the UDDI registry is called Publish and would be generally done by Provider.**
➔ **Now the consumer has to connect to the registry and performs a search on the registry which is called Discovery to find an appropriate WSDL document.**
➔ Downloads it locally and now starts understanding the information about provider
➔By the above we understood the Web-Service Architecture & the language for exchanging the information is XML. Binding protocol is SOAP which is nothing but XML and Description of the Service is WSDL, is also XML. Registry in which we store the WSDL is UDDI, built on top of XML technologies, everything in Web service is built on XML and without XML Web Services would not be possible.

XML:

➔ **XML stands for Extensible Markup Language. Markup language is the language Using which you can build other languages like, HTML, XML.**
➔ XML is defined and governed by W3c Organization.
➔ The first and final version of XML is XML 1.0.
➔ XML is the document which store & represents data.
➔ XML is not a programming language.
➔ In XML there are no keywords or reserved words. What you write will become the element of that XML document.

XML Element:

➔ We cannot store the data in an XML in any orbitary manner. The data we are trying to store inside the XML must and should be enclosed in between the tags. These tags are two types.

➔ Element in an XML is written in angular braces for e.g. <tag>. In XML there are Two types of elements as follows.

➔ **Start Element/Opening Tag: -** Start element is the element which is written in <element-Name> indicating the start of a block.

➔ **End Element/End Tag: -** End element is the element which is written in </element-Name> indicating the end of a block.

➔ As everything is written in terms of start and end elements, XML is said to be more Structured in nature. An XML Element may contain content or may contain other Child elements under it.

➔ XML elements which contain other elements in it or child elements within it are called as **Compound elements** or XML Containers.

➔ An XML element which directly contains data or content (simple text data) within the starting & ending tag is called **Simple elements**.

Well-formness:

➔ Well-formness indicates the readability nature of an XML document. If an XML document is said to be well-formed then it is readable in nature.

Rules For Well-formness:

➔ Every XML document must start with PROLOG.

Prolog :

Prolog stands for processing instruction, and typically used for understanding about the version of XML used and the data encoding used.

Example: - <?xml version="1.0" encoding="utf-8"?>

Root Element:

➔XML document must contain a root element, and should be the only one root element. All the other elements should be the children of root element.

➔ As XML stores the in **hierarchical representation**, so there should be must one & only one root element.

Level constraint:

➔Every start element must have an end element and the level at which you open a start element, the same level you need to close your end element as well.

Validity:

➔ Validity of the XML document would be defined by the application which is going to process your XML document. Let's consider a scenario as follows.

Example:

➔ You want to get driving license. In order to get a driving license you need to Follow certain process like filling the RTA forms and signing them and submitting To the RTA department.

➔ Instead of this can you write your own format of letter requesting the driving license from an RTA department? Which seems to be not relevant because driving license is something that would be issued by the RTA department.

➔So, the whole and sole authority of defining what should a person has to provide data to get a driving license will lies in the hands of RTA department rather than you.

➔ In the same way when an application is going to process your xml document, the Authority of defining what should be there as part of that xml is lies in the hands of the application which is going to process your document.

Example:- Purchase Order xml document.

```
<purchase-order>
    <order-item>
        <item>
            <item-code>c121</item-code>
            <quantity>34</quantity>
        </item>
    </order-item>
    <shipping-address>
        <address-line1>544</address-line1>
        <address-line2>mytrivanam</address-line2>
        <city>Hyderabad</city>
        <state>ts</state>
        <zip>700058</zip>
        <country>india</country>
    </shipping-address>
</purchase-order>
```

➔ In the above xml even though it confirms to all the well-formness rules, it cannot be used for business transaction, **as the 2nd <quantity> element carries the data as "abc"** which doesn't makes any sense.
➔ So, in order to check for data validity we need to define the validation criteria of an XML document in either a DTD or XSD document

XML Attribute:

➔ **If we want to have supplementary information attach to an element**, instead of having it as content or another element, we can write it as an Attribute of the element.

```
<address type="Office">
    <address-line1>505</address-line1>
    <address-line2>Mytrivanam</address-line2>
    <city>Hyderabad</city>
    <state>Telengana</state>
    <zip>700058</zip>
    <country>India</country>
</address>
```

In the above example "address" is an element which contains one attributes name Which acts as an supplementary information.

DTD

DTD

➔ DTD **stands for Document Type Definition.** It is the document which defines the structure of an XML document.
➔ DTD are governed & owned by W3c organization.

Who defines the DTD documents?

➔ The person who wants to read the XML or who wants to receive the XML in a specific structure has to define the structure of the XML inside the DTD.
➔DTD are not the XML documents, but those are used for defining the structure of XML documents.

How to write a DTD document or How do I need to define a structure of an XML inside the DTD document?

➔ As we have two types of XML elements, element which carry data are called **Simple element** and an element which contain child-element is called as **Compound type**.

How to declare simple elements in DTD

```
<!Element    ElementName    (CONTENT-TYPE OR CONTENT-MODEL) >
```

Ex:

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

➔Here I declared an element, it means my xml should contain an element whose name is **country and** it contains PCDATA (**Parseable character data)**.
➔In an XML character like '<', '>', ' " ' , " ' " and '&' are valid character and if the text contains those characters the parser are going to parse them and also will expand for extra entries.

How to declare Compound element in DTD

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT purchase-order (order-item, shipping-address)>
    <!ELEMENT order-item (item)>
        <!ELEMENT item (item-code,quantity)>
            <!ELEMENT item-code (#PCDATA)>
            <!ELEMENT quantity (#PCDATA)>
    <!ELEMENT shipping-address (address-line1,address-line2,city,state,zip,country)>
        <!ELEMENT address-line1 (#PCDATA)>
        <!ELEMENT address-line2 (#PCDATA)>
        <!ELEMENT city (#PCDATA)>
        <!ELEMENT state (#PCDATA)>
        <!ELEMENT zip (#PCDATA)>
        <!ELEMENT country (#PCDATA)>
```

Rules to declaring element in DTD

➔While writing DTD we need to follow certain rules or conventions while declaring elements in DTD:

➔ Element name should follow the java variable naming conventions.

➔ Content of an XML is always case-sensitive.

➔ So if we declare element names in small case in DTD, it should appear in small case in XML as well.

➔ The child elements separator in the above example is used as "," comma. This is also called as sequence separator.

How to write DTD

➔Identify all the simple element of XML from bottom of the XML and write them in DTD.

➔Identify compound element of XML from lowest level in the XML and write them in the DTD.

Example-1

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE purchase-order SYSTEM "E:\webservices\po.dtd">
<purchase-order>
        <order-item>
                <item>
                        <item-code></item-code>
                        <quantity></quantity>
                </item>
        </order-item>
        <shipping-address>
                <address-line1></address-line1>
                <address-line2></address-line2>
                <city></city>
                <state></state>
                <zip></zip>
                <country></country>
        </shipping-address>
</purchase-order>
```

DTD for above XML

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT purchase-order (order-item, shipping-address)>

<!ELEMENT order-item (item)>

<!ELEMENT item (item-code,quantity)>

<!ELEMENT item-code (#PCDATA)>

<!ELEMENT quantity (#PCDATA)>

<!ELEMENT shipping-address (address-line1,address-line2,city,state,zip,country)>

<!ELEMENT address-line1 (#PCDATA)>

<!ELEMENT address-line2 (#PCDATA)>

<!ELEMENT city (#PCDATA)>

<!ELEMENT state (#PCDATA)>

<!ELEMENT zip (#PCDATA)>

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

Occurrences of an Element under another element

➔In above xml if you observe the order-item element can contain any number of item elements in it.
➔But aleast one item element must be there for a purchase-order this is called occurrence of an element under another element.
➔To achieve occurrence we have to use following symbols.

⇨'?' child element can appear zero or one time (0 | 1).

⇨'+' child element must appear at least once and can repeat any number of time (1 | N).

⇨'*' child element is optional and can appear any number of time.

Example

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

<!ELEMENT order-item (item+)>

<!ELEMENT item (item-code,quantity)>

<!ELEMENT item-code (#PCDATA)>

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

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE purchase-order SYSTEM "E:\webservices\po.dtd">
<order-item>
    <item>
        <item-code></item-code>
        <quantity></quantity>
    </item>
    <item>
        <item-code></item-code>
        <quantity></quantity>
    </item>
</order-item>
```

Element with ANY Content

➔Element declare with the content type as ANY, can contain any element of XSD in any Order.

➔Here in 1st Example we can't change the element order and can't skip any element.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE mail-body SYSTEM "E:\webservices\any.dtd">
<mail-body>
    <to>mail4dhananjaya@gmail.com</to>
    <from>javatech.dj@gmail.com</from>
    <heading>ANY USE CASE</heading>
    <body>NA</body>
</mail-body>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT mail-body (to,from,heading,body)>
<!ELEMENT to (#PCDATA)>
<!ELEMENT from (#PCDATA)>
<!ELEMENT heading (#PCDATA)>
<!ELEMENT body (#PCDATA)>
```

➔But when we are using ANY content type as part of our element the any element tag can allow in any order even zero child elements also allowed and we can skip any element there is no problem.

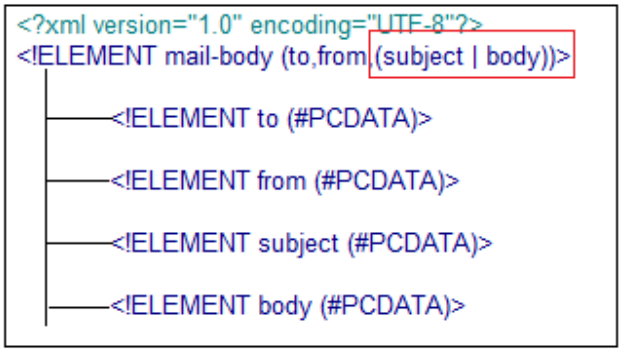
➔But element should be declared inside XSD those are only allowed.

```
<?xml version="1.0" encoding="UTF-8"?>
<!--<!ELEMENT mail-body (to,from,heading,body)>-->

<!ELEMENT mail-body ANY>
    |-- <!ELEMENT to (#PCDATA)>
    |-- <!ELEMENT from (#PCDATA)>
    |-- <!ELEMENT heading (#PCDATA)>
    |-- <!ELEMENT body (#PCDATA)>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE mail-body SYSTEM "E:\webservices\any.dtd">
<mail-body>
    <to></to>
    <from></from>
    <!--<heading></heading>-->
     <body></body>
</mail-body>
```

Element with Either | OR content

➔If I want to do either <subject >or <body> then I can declare Compound element like.




```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE mail-body SYSTEM "E:\webservices\any.dtd">
<mail-body>
        <to></to>
        <from></from>
        <subject></subject>
        <!--<body></body>-->
</mail-body>
```

Element with Mixed content

We can declare the content with mixed of parssable data and element.

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT mail-body (#PCDATA|to | from | subject | body)*>
<!ELEMENT to (#PCDATA)>
<!ELEMENT from (#PCDATA)>
<!ELEMENT subject (#PCDATA)>
<!ELEMENT body (#PCDATA)>
```

* is required if we are declaring #PCDATA

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE mail-body SYSTEM "E:\webservices\any.dtd">
<mail-body>
        <to></to>
        <!--<from></from>
        <subject></subject>
        <body></body>-->
</mail-body>
```

Declaring Attribute for an Element

➔If I want to give some additional information for the element then I have to use attribute.

CDATA

Type	Description
CDATA	The value is character data

Value	Description
#REQUIRED	The attribute is mandatory

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

<!ELEMENT bean (property | constructor-arg)>
        <!ATTLIST  bean id CDATA #REQUIRED>
        <!ATTLIST  bean class CDATA #REQUIRED>

            <!ELEMENT property (#PCDATA)>
            <!ATTLIST  property name CDATA #REQUIRED>
            <!ATTLIST  property value  CDATA #REQUIRED>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE bean SYSTEM "E:\webservices\bean.dtd">
<bean id="" class="">
        <property name="" value=""></property>
</bean>
```

ENUMERATION

Type	Description
(**ENUMERATION-1 | ENUMERATION-2 | ...**)	The value must be one from the enumerated list of values

Value	Description
#REQUIRED	The attribute is mandatory

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE shipping-address SYSTEM "E:\webservices\po1.dtd">
<shipping-address type="TEMPORARY
                        PERMANENT">
        <address-line1></address-line1>
        <address-line2></address-line2>
        <city></city>
        <state></state>
        <zip></zip>
        <country></country>
</shipping-address>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT shipping-address (address-line1,address-line2,city,state,zip,country)>
    <!ATTLIST shipping-address type (PERMANENT | TEMPORARY) #REQUIRED>

    <!ELEMENT address-line1 (#PCDATA)>

    <!ELEMENT address-line2 (#PCDATA)>

    <!ELEMENT city (#PCDATA)>

    <!ELEMENT state (#PCDATA)>

    <!ELEMENT zip (#PCDATA)>

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

ID

Type	Description
ID	The value is unique ID
IDREF	The value is ID of another element

Value	Description
#REQUIRED	The attribute is mandatory

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

<!ELEMENT beans (bean+)>
    <!ELEMENT bean (property | constructor-arg)>
        <!ATTLIST  bean id ID #REQUIRED>
        <!ATTLIST  bean class CDATA #REQUIRED>

    <!ELEMENT property (#PCDATA)>
        <!ATTLIST  property name CDATA #REQUIRED>
        <!ATTLIST  property ref  IDREF #REQUIRED>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE beans SYSTEM "E:\webservices\bean.dtd">
<beans>
        <bean class="com.po.PurchaseOrder" id="purchaseOrder">
                <property name="po" ref="purchaseOrder"/>
        </bean>
</beans>
```

NMTOKEN , DEFAULT VALUE

Type	Description
NMTOKEN	➔ NMTOKEN specifies the any string character exclude whitespace. Before and after whitespace automatic trimmed. ➔NMTOKEN not allowed whitespaces in middle of character

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT empinfo (employee+)>
    <!ELEMENT employee (name, designation)>
        <!ELEMENT name (#PCDATA)>
        <!ELEMENT designation (#PCDATA)>
            <!ATTLIST designation discipline NMTOKEN "web_developer">
```

default value to attribute

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE empinfo SYSTEM "E:\webservices\emp.dtd">
<empinfo>
    <employee>
        <name>Dhananjaya</name>
        <designation discipline="Software_developer">Administrator</designation>

        <!--<designation discipline="Software developer">Administrator</designation>-->

    </employee>

    <employee>
        <name>Dhananjaya</name>
        <designation>Administrator</designation> // web_developer
    </employee>
</empinfo>
```

When i am using CDATA it is valid

but when i am using NMTOKEN it is INVALID

because NMTOKEN not allowing white spaces

When i am not puting any discipline automatically default value will consider as input

#REQUIRED , #IMPLIED , #FIXED

#REQUIRED

➔When I am using #REQUIRED in DTD then value of **type attribute is mandatory in shipping address element.**

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT shipping-address (address-line1,address-line2,city,state,zip,country)>
    <!ATTLIST shipping-address type CDATA #REQUIRED>
    <!ELEMENT address-line1 (#PCDATA)>
    <!ELEMENT address-line2 (#PCDATA)>
    <!ELEMENT city (#PCDATA)>
    <!ELEMENT state (#PCDATA)>
    <!ELEMENT zip (#PCDATA)>
    <!ELEMENT country (#PCDATA)>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE shipping-address SYSTEM "E:\webservices\po1.dtd">
<shipping-address type="PERMANENT">
    <address-line1>505</address-line1>
    <address-line2>Mytrivanam</address-line2>
    <city>Hyderabad</city>
    <state>Telengana</state>
    <zip>700038</zip>
    <country>INDIA</country>
</shipping-address>
```

Mandatory

#IMPLIED

➔When I am using #IMPLIED in DTD then **type attribute itself is optional in shipping address element.**

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT shipping-address (address-line1,address-line2,city,state,zip,country)>
    <!ATTLIST shipping-address type CDATA #IMPLIED>
    <!ELEMENT address-line1 (#PCDATA)>
    <!ELEMENT address-line2 (#PCDATA)>
    <!ELEMENT city (#PCDATA)>
    <!ELEMENT state (#PCDATA)>
    <!ELEMENT zip (#PCDATA)>
    <!ELEMENT country (#PCDATA)>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE shipping-address SYSTEM "E:\webservices\po1.dtd">
<shipping-address type="">
    <address-line1>505</address-line1>
    <address-line2>Mytrivanam</address-line2>
    <city>Hyderabad</city>
    <state>Telengana</state>
    <zip>700038</zip>
    <country>INDIA</country>
</shipping-address>
```

Optional

#FIXED

➔When I am using #FIXED in DTD then the **value of type attribute** **is fixed in shipping address element we can't change its value.**

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT shipping-address (address-line1,address-line2,city,state,zip,country)>
        <!ATTLIST shipping-address type CDATA #FIXED "PERMANENT">
    <!ELEMENT address-line1 (#PCDATA)>
    <!ELEMENT address-line2 (#PCDATA)>
    <!ELEMENT city (#PCDATA)>
    <!ELEMENT state (#PCDATA)>
    <!ELEMENT zip (#PCDATA)>
    <!ELEMENT country (#PCDATA)>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE shipping-address SYSTEM "E:\webservices\po1.dtd">
<shipping-address type="PERMANENT">
        <address-line1>505</address-line1>
        <address-line2>Mytrivanam</address-line2>
        <city>Hyderabad</city>
        <state>Telengana</state>
        <zip>700038</zip>
        <country>INDIA</country>
</shipping-address>
```

WE CAN'T CHANGE

Drawbacks of DTD

➔**DTD's are not type safe,** when we declare a simple element, we indicate it should contain data of (#PCDATA).

➔#PCDATA means parsable character data means any data that computer represented format.

➔ Means it allows any type int, float, string.

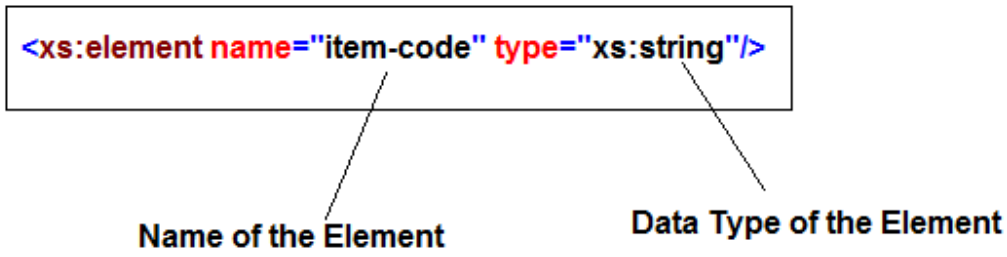
➔This is the limitation with DTD document.

XSD

XML Schema Document (XSD)

➔XSD Stands for XML schema document.XSD is also an XML document .It is owned by W3C Organization, The latest version of XSD is 1.1.
➔XSD used for defining the structure of an XML document and validating the content of XML.
➔Even though we can use DTD's for defining the validity of an XML document, it is not type safe and it is not flexible.
➔XSD is more powerful and is type strict in nature.
➔XSD root element is schema and it starts with prolog.

How to represent simple element in XSD?



How to represent compound element in XSD?

➔In order to represent compound element of an XML document in an XSD, first we need to create a data type declaration representing structure of that xml element.
➔Then we need to declare element of that user-defined type.

```
<xs:element name="item" type="to:item-type"/>
<xs:complexType name="item-type">
        <xs:sequence>
                <xs:element name="item-code" type="xs:string"/>
                <xs:element name="quantity" type="xs:int"/>
        </xs:sequence>
</xs:complexType>
```

In order to create user-defined data type in XSD document you need to declare a complex type.

Approach -1

➔It is like declaring a class type in java .in java we are declaring user-defined data-type using class-declaration.

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

    <xs:element name="purchase-order" type="purchase-order-type"/>

    <xs:complexType name="purchase-order-type">
        <xs:sequence>
            <xs:element name="order-item" type="order-item-type"/>
        </xs:sequence>
    </xs:complexType>

    <xs:complexType name="order-item-type">
        <xs:sequence>
            <xs:element name="item" type="item-type"/>
        </xs:sequence>
    </xs:complexType>

    <xs:complexType name="item-type">
        <xs:sequence>
            <xs:element name="item-code" type="xs:string"/>
            <xs:element name="quantity" type="xs:int"/>
        </xs:sequence>
    </xs:complexType>
</xs:schema>
```

Let's take an XML document for which represents the structure.

```
<?xml version="1.0" encoding="UTF-8"?>
<purchase-order xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                xsi:noNamespaceSchemaLocation="file:///E:/webservices/po.xsd">
    <order-item>
        <item>
            <item-code>C0122</item-code>
            <quantity>57</quantity>
        </item>
    </order-item>
</purchase-order>
```

Approach -2

➔In This approach it's like an inner class declaration in java

➔In this approach no need to declare complex element (data type) outside.

XSD

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
attributeFormDefault="unqualified">
    <xs:element name="purchase-order">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="order-item">
                    <xs:complexType>
                        <xs:sequence>
                            <xs:element name="item">
                                <xs:complexType>
                                    <xs:sequence>
                                        <xs:element name="item-code" type="xs:string"/>
                                        <xs:element name="quantity" type="xs:int"/>
                                    </xs:sequence>
                                </xs:complexType>
                            </xs:element>
                        </xs:sequence>
                    </xs:complexType>
                </xs:element>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
</xs:schema>
```

XML

```
<?xml version="1.0" encoding="UTF-8"?>
<purchase-order xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                xsi:noNamespaceSchemaLocation="file:///E:/webservices/po.xsd">
    <order-item>
        <item>
            <item-code>C0122</item-code>
            <quantity>57</quantity>
        </item>
    </order-item>
</purchase-order>
```

Sequence vs All

➔We are using **<xs:sequence> and <xs:all>** in side complex type.

When should we go for <xs:sequence> when should we go for <xs:all> ?

➔If our requirement is which are the elements declared in **<xs:complexType>,** all the elements **must appear in same order in XML document.**

XSD

```
<xs:complexType name="item-type">
        <xs:sequence>
                <xs:element name="item-code" type="xs:string"/>
                <xs:element name="quantity" type="xs:int"/>
        </xs:sequence>
</xs:complexType>
```

XML

```
<item>
      <item-code>C0122</item-code>
      <quantity>57</quantity>
</item>
```

➔When we use **<xs:all> under** item element **<item-code> and <quantity> may not appear in same order.**

```
<xs:complexType name="item-type">
        <xs:all>
                <xs:element name="item-code" type="xs:string"/>
                <xs:element name="quantity" type="xs:int"/>
        </xs:all>
</xs:complexType>
```

```
<item>
  <quantity>57</quantity>
  <item-code>C0122</item-code>
</item>
```

Extending ComplexType

➔XSD allow extending user defined data type by the means of inheritance.

PARENT DATA TYPE

```
<xs:complexType name="shipping-address">
        <xs:sequence>
                <xs:element name="address-line1" type="xs:string"/>
                <xs:element name="address-line2" type="xs:string"/>
                <xs:element name="city" type="xs:string"/>
                <xs:element name="state" type="xs:string"/>
                <xs:element name="zip" type="xs:string"/>
                <xs:element name="country" type="xs:string"/>
        </xs:sequence>
</xs:complexType>
```

CHILD DATA TYPE

```
<xs:complexType name="us-shipping-address-type">
        <xs:complexContent>
                <xs:extension base="shipping-address">
                        <xs:sequence>
                                <xs:element name="county-code" type="xs:string"/>
                        </xs:sequence>
                </xs:extension>
        </xs:complexContent>
</xs:complexType>
```

Inheriting from parent

```
<purchase-order xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="file:///E:/webservices/Untitled8.xsd">
        <us-shipping-address>
                <address-line1>505</address-line1>
                <address-line2>DC Building</address-line2>
                <city>Bosten</city>
                <state>Bosten</state>
                <zip>547881</zip>
                <country>US</country>
                <county-code>566690</county-code>
        </us-shipping-address>
</purchase-order>
```

Declaring Attribute for Complex Type

➔In XML we can place additional information to an element in an attribute.

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

        <xs:element name="shipping-address" type="shipping-address-type"/>
        <xs:complexType name="shipping-address-type">
                <xs:sequence>
                        <xs:element name="address-line1" type="xs:string"/>
                        <xs:element name="address-line2" type="xs:string"/>
                        <xs:element name="city" type="xs:string"/>
                        <xs:element name="state" type="xs:string"/>
                        <xs:element name="zip" type="xs:string"/>
                        <xs:element name="country" type="xs:string"/>
                </xs:sequence>

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

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

➔We should declare a attribute in a complexType at the last after <xs:sequence> tag because sequence doesn't apply for attributes.

```
<?xml version="1.0" encoding="UTF-8"?>
<shipping-address type="PERMANENT" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                                   xsi:noNamespaceSchemaLocation="file:///E:/webservices/po3.xsd">
        <address-line1>C-16</address-line1>
        <address-line2>N.K. Nagar</address-line2>
        <city>Rourkela</city>
        <state>Odisha</state>
        <zip>769016</zip>
        <country>India</country>
</shipping-address>
```

Declaring Attribute for Simple Elements

➔To declare an attribute for a simple element we have to use a complexType (data type) inside complexType we have to add-on an attribute to a simpleType element.

Let's see how it has declared

```
<zip zone="local">769016</zip>
```

```
<xs:element name="zip" type="zip-type"/>

<xs:complexType name="zip-type">
        <xs:simpleContent>
                <xs:extension base="xs:string">
                        <xs:attribute name="zone" type="xs:string" use="required"/>
                </xs:extension>
        </xs:simpleContent>
</xs:complexType>
```

➔In this example <zip> is a simple element so to extend an attribute into it.
➔So we have to follow some guide lines
- ➔First we have to see <zip> tag is a complexType or simpleType.
- ➔If it is simpleType then we have to check any complexType for zip has declared or not.
- ➔If not then first we have to take a <simpleContent> inside <complexType> because we are extending simple type functionality.
- ➔After that we have to see what the data type of attribute.
- ➔Suppose data type is xs:string , then we have to extend xs:string means we have to use <xs:extension base="xs:string"> and add a attribute in it.
- ➔After that we have to give the use means is it optional/required/prohibited

Working with choice

➔Suppose my requirement is some time shipping-address and some time us-shipping-address both are not required at a time.
➔In this case I have to use <xs:choice> instead of <xs:sequence>

➔It is not necessary to present both but either shipping-address or us-shipping-address is mandatory. To impose (force) such type of constraints we need to use <xs:choice> element.

```
<xs:element name="purchase-order" type="purchase-order-type"/>
<xs:complexType name="purchase-order-type">
        <xs:choice>
                <xs:element name="shipping-address" type="shipping-address-type"/>
                <xs:element name="us-shipping-address" type="us-shipping-address-type"/>
        </xs:choice>
</xs:complexType>
```

PARENT DATA TYPE

```
<xs:complexType name="shipping-address">
        <xs:sequence>
                <xs:element name="address-line1" type="xs:string"/>
                <xs:element name="address-line2" type="xs:string"/>
                <xs:element name="city" type="xs:string"/>
                <xs:element name="state" type="xs:string"/>
                <xs:element name="zip" type="xs:string"/>
                <xs:element name="country" type="xs:string"/>
        </xs:sequence>
</xs:complexType>
```

CHILD DATA TYPE

```
<xs:complexType name="us-shipping-address-type">
        <xs:complexContent>
                <xs:extension base="shipping-address">
                        <xs:sequence>
                                <xs:element name="county-code" type="xs:string"/>
                        </xs:sequence>
                </xs:extension>
        </xs:complexContent>
</xs:complexType>
```

Using Ref

➔If the same element is used in multiple complexType then instead of declaring each and every complexType we can declare it as globally(directly in side <xs:schema>) and can reuse that element by using its **ref** key word.

```
<xs:element name="item-code" type="xs:string"/>
<xs:element name="quantity" type="xs:int"/>
```

```
<xs:complexType name="order-type">
        <xs:sequence>
                <xs:element ref="item-code"/>
                <xs:element  ref="quantity"/>
        </xs:sequence>
</xs:complexType>
```

```
<xs:complexType name="bill-type">
        <xs:sequence>
                <xs:element ref="item-code"/>
                <xs:element  ref="quantity"/>
        </xs:sequence>
</xs:complexType>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<purchase-type xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="file:///E:/webservices/po45.xsd">
        <order>
                <item-code>C8995</item-code>
                <quantity>46</quantity>
        </order>
        <bill>
                <item-code>C8995</item-code>
                <quantity>46</quantity>
        </bill>
</purchase-type>
```

Group of element Using Ref

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


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

        <xs:group name="address-tamplate">
                <xs:sequence>
                        <xs:element name="address-line1" type="xs:string"/>
                        <xs:element name="address-line2" type="xs:string"/>
                        <xs:element name="city" type="xs:string"/>
                        <xs:element name="state" type="xs:string"/>
                        <xs:element name="zip" type="xs:string"/>
                        <xs:element name="country" type="xs:string"/>
                </xs:sequence>
        </xs:group>


        <xs:complexType name="shipping-address-type">
                        <xs:group ref="address-tamplate"/>
        </xs:complexType>


</xs:schema>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<shipping-address xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="file:///E:/webservices/po.xsd">
        <address-line1>C-16</address-line1>
        <address-line2>N.K Nagar</address-line2>
        <city>Rourkela</city>
        <state>Odisha</state>
        <zip>769016</zip>
        <country>India</country>
</shipping-address>
```

Restricting Simple Type

➔XSD allow defining data validation on the element data.
➔To achieve this validation we have to use our own simpleType by extending pre-defined data types and can impose (force) restriction on it.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" ">
        <xs:element name="shipping-address" type="shipping-address-type"/>
        <xs:complexType name="shipping-address-type">
                <xs:sequence>
                        <xs:element name="zip" type="zip-type"/>
                </xs:sequence>
        </xs:complexType>
        <xs:simpleType name="zip-type">
                <xs:restriction base="xs:int">
                        <xs:minInclusive value="700000"/>
                        <xs:maxInclusive value="700050"/>
                </xs:restriction>
        </xs:simpleType>
</xs:schema>
```

```
<shipping-address>
        <zip>70005</zip>
</shipping-address>
```

allowed value between 700000 to 700050

Enumeration for simple Element

```
<?xml version="1.0" encoding="UTF-8"?>
<shipping-address xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="file:///E:/webservices/zip.xsd">
        <address-line1>C-16</address-line1>
        <address-line2>N.K.Nagar</address-line2>
        <city>Rourkela</city>
        <state>Odisha</state>
        <zip>769016</zip>
        <country>INDIA</country>
                 US
</shipping-address>
```

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

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

        <xs:complexType name="shipping-address-type">
                <xs:sequence>
                        <xs:element name="address-line1" type="xs:string"/>
                        <xs:element name="address-line2" type="xs:string"/>
                        <xs:element name="city" type="xs:string"/>
                        <xs:element name="state" type="xs:string"/>
                        <xs:element name="zip" type="xs:string"/>
                        <xs:element name="country" type="country-type"/>
                </xs:sequence>
        </xs:complexType>

        <xs:simpleType name="country-type">
                <xs:restriction base="xs:string">
                        <xs:enumeration value="INDIA"/>
                        <xs:enumeration value="US"/>
                </xs:restriction>
        </xs:simpleType>
</xs:schema>
```

Enumeration for Attributes

```
<?xml version="1.0" encoding="UTF-8"?>
<shipping-address xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="file:///E:/webservices/zip.xsd">
        <address-line1>C-16</address-line1>
        <address-line2>N.K Nagar</address-line2>
        <city>Rourkela</city>
        <state>Odisha</state>
        <zip zone="LOCAL"></zip>
                    OUT SIDE
        <country>India</country>
</shipping-address>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
        <xs:element name="shipping-address" type="shipping-address-type"/>
        <xs:complexType name="shipping-address-type">
                <xs:sequence>
                        <xs:element name="address-line1" type="xs:string"/>
                        <xs:element name="address-line2" type="xs:string"/>
                        <xs:element name="city" type="xs:string"/>
                        <xs:element name="state" type="xs:string"/>
                        <xs:element name="zip" type="zip-type"/>
                        <xs:element name="country" type="xs:string"/>
                </xs:sequence>
        </xs:complexType>

        <xs:complexType name="zip-type">
                <xs:simpleContent>
                        <xs:extension base="xs:string">
                                <xs:attribute name="zone" type="zone-enum" use="required"/>
                        </xs:extension>
                </xs:simpleContent>
        </xs:complexType>

        <xs:simpleType name="zone-enum">
                <xs:restriction base="xs:string">
                        <xs:enumeration value="LOCAL"/>
                        <xs:enumeration value="OUT SIDE"/>
                </xs:restriction>
        </xs:simpleType>

</xs:schema>
```

Occurrences in XSD

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

        <xs:element name="order-item" type="order-item-type"/>
        <xs:complexType name="order-item-type">
                <xs:sequence>
                        <xs:element name="item" type="item-type" minOccurs="1" maxOccurs="unbounded"/>
                </xs:sequence>
        </xs:complexType>
        <xs:complexType name="item-type">
                <xs:sequence>
                        <xs:element name="item-code"/>
                        <xs:element name="quantity"/>
                </xs:sequence>
        </xs:complexType>
</xs:schema>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<order-item xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
            xsi:noNamespaceSchemaLocation="file:///E:/webservices/orderItem.xsd">
        <item>
                <item-code>C121</item-code>
                <quantity>45</quantity>
        </item>
        <item>
                <item-code>B45</item-code>
                <quantity>8</quantity>
        </item>
        <item>
                <item-code>H3TH</item-code>
                <quantity>27</quantity>
        </item>
</order-item>
```

Occurrences of Group in sequentially and repeatedly

```
<?xml version="1.0" encoding="UTF-8"?>
<purchase xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xsi:noNamespaceSchemaLocation="file:///E:/webservices/po45.xsd">
    <order>
        <item-code>C301</item-code>
        <quantity>13</quantity>
    </order>
    <bill>
        <item-code>C301</item-code>
        <quantity>13</quantity>
    </bill>
    <order>
        <item-code>H13M</item-code>
        <quantity>50</quantity>
    </order>
    <bill>
        <item-code>H13M</item-code>
        <quantity>50</quantity>
    </bill>
</purchase>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:element name="purchase" type="purchase-type"/>
    <xs:complexType name="purchase-type">
        <xs:sequence minOccurs="1" maxOccurs="unbounded ">
            <xs:element name="order" type="order-type"/>
            <xs:element name="bill" type="bill-type"/>
        </xs:sequence>
    </xs:complexType>

    <xs:complexType name="order-type">
        <xs:sequence>
    <xs:element name="item-code" type="xs:string"/>
    <xs:element name="quantity" type="xs:int"/>
        </xs:sequence>
    </xs:complexType>

    <xs:complexType name="bill-type">
        <xs:sequence>
    <xs:element name="item-code" type="xs:string"/>
    <xs:element name="quantity" type="xs:int"/>
        </xs:sequence>
    </xs:complexType>
</xs:schema>
```

Occurrences of Element any time but total group in sequentially

```
<xs:complexType name="purchase-type">
        <xs:sequence>
                <xs:element name="order" type="order-type" minOccurs="1" maxOccurs="unbounded "/>
                <xs:element name="bill" type="bill-type" minOccurs="1" maxOccurs="unbounded "/>
        </xs:sequence>
</xs:complexType>
```

```
<purchase>
        <order></order>
        <order></order>
        <bill></bill>
        <bill></bill>
</purchase>
```

Occurrences of any of the one element and can be occurred multiple time.

```
<xs:complexType name="purchase-type">
        <xs:choice>
                <xs:element name="order" type="order-type" minOccurs="1" maxOccurs="unbounded "/>
                <xs:element name="bill" type="bill-type" minOccurs="1" maxOccurs="unbounded "/>
        </xs:choice>
</xs:complexType>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<purchase xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
            xsi:noNamespaceSchemaLocation="file:///E:/webservices/po45.xsd">
        <order></order>
        <order></order>
</purchase>
```

Occurrences of any element and can be occurred either 0 or 1 in any order.

```
<xs:complexType name="purchase-type">
        <xs:all>                                                    // if  we are using <xs:all> By default minOccurs="0" maxOccurs="1"
                <xs:element name="order" type="order-type" minOccurs="0" maxOccurs="1"/>
                <xs:element name="bill" type="bill-type" />
        </xs:all>
</xs:complexType>
```

```
<purchase>
        <bill></bill>
        <order></order>
</purchase>
```

```
<purchase>
        <order></order>
        <bill></bill>
</purchase>
```

Occurrences of any element and can be occurred multiple time in any order.

```
<xs:complexType name="purchase-type">
        <xs:choice minOccurs="1" maxOccurs="unbounded">
                <xs:element name="order" type="order-type" />
                <xs:element name="bill" type="bill-type" />
        </xs:choice>
</xs:complexType>
```

```
<purchase>
        <bill></bill>
        <bill></bill>
        <bill></bill>
        <order></order>
        <order></order>
</purchase>
```

Namespaces

➔In XSD Namespaces as like package in java.
➔Namespace is used to resolve naming collision between there complexTypes.
➔ In java it will allow you to uniquely identify a type by prefixing the package name.
➔ In XSD it allows you to resolve the type naming conflicts by means of Namespace declaration.

XSD Namespaces has two faces

➔ Declaring the namespace in the XSD document using targetNameSpace declaration
➔ Using the elements that are declared under a namespace in XML document.

XSD targetNameSpace

➔TargetNameSpace declaration is similar to a package declaration in java.
➔While using them you will refer with fully qualified name.
➔Similarly when you create a complexType or an Element you will bind them to a namespace
➔So that while referring them you need to use Qualified Name
➔To declare a namespace in XSD we need to use targetNameSpace.
➔ So, while creating an element "item-type" of type courseType you should use the Qualified Name of the item-type rather simple name. (Qualified Name means namespace:element/typeName).
➔ But the namespace labels could be any of the characters in length,
➔So if we prefix the entire namespace label to element/type names it would become hard to read.
➔So, to avoid this problem XSD has introduced a concept called short name.
➔Instead of referring to namespace labels you can define a short name for that namespace label using **xmlns** declaration at the <schema> level and you can use the short name as prefix instead of the complete namespace label as shown below.

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

    <xs:element name="purchase-order" type="est:purchase-order-type"/>

    <xs:complexType name="purchase-order-type">
        <xs:sequence>
            <xs:element name="item" type="est:item-type"/> // Using complexType
        </xs:sequence>
    </xs:complexType>

    <xs:complexType name="item-type">                    // Declaring ComplexType
        <xs:sequence>
            <xs:element name="item-code" type="xs:string"/>
            <xs:element name="quantity" type="xs:int"/>
        </xs:sequence>
    </xs:complexType>

</xs:schema>
```

Using elements from an xml namespace (xmlns)

➔ While writing an XML document, you need to link it to XSD to indicate it is following the structure defined in that XSD.

➔In order to link an XML to an XSD we use an attribute “schemaLocation”.

➔ “schemaLocation” attribute declaration has two pieces of information. First part is representing the namespace from which you are using the elements. Second part is the Path of XSD which contains this namespace.

➔ But the namespace labels can be arbitrary string of characters in any length. So, we need to define short name, so that we can prefix short name while referring the element.

```
<?xml version="1.0" encoding="UTF-8"?>
<est:purchase-order xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="com.ebay.sales.types file:///E:/webservices/namespace1.xsd" xmlns:est="com.ebay.sales.types">
        <item>
                <item-code>C121</item-code>
                <quantity>46</quantity>
        </item>
</est:purchase-order>
```

Namespace

Location of XSD

Short Name

➔If you want to include two XSD documents in the xml then you need to declare in schemaLocation tag

<namespace1> <schemalocation1> <namespace2> <schemalocation2>.

```
<?xml version="1.0" encoding="UTF-8"?>
<purchase-order xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="com.ebay.sales.types  file:///E:/webservices/namespace1.xsd
                    com.ebay.purchase.types  file:///E:/webservices/namespace2.xsd ">
```

If elemenetFormDefault is unqualified

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

    <xs:element name="purchase-order" type="est:purchase-order-type"/>
    <xs:complexType name="purchase-order-type">
        <xs:sequence>
            <xs:element name="item" type="est:item-type"/>
        </xs:sequence>
    </xs:complexType>

    <xs:complexType name="item-type">
        <xs:sequence>
            <xs:element name="item-code" type="xs:string"/>
            <xs:element name="quantity" type="xs:int"/>
        </xs:sequence>
    </xs:complexType>

</xs:schema>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<st:purchase-order xmlns:st="com.ebay.sales.types"
                   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                   xsi:schemaLocation="com.ebay.sales.types file:///E:/webservices/namespace1.xsd">
    <item>
        <item-code>C54</item-code>
        <quantity>30</quantity>
    </item>
</st:purchase-order>
```

If in XSD elementFormDefault is unqualified then in XML no need to prefix any short name

If elemenetFormDefault is qualified (by default)

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

    <xs:element name="purchase-order" type="est:purchase-order-type"/>
    <xs:complexType name="purchase-order-type">
        <xs:sequence>
            <xs:element name="item" type="est:item-type"/>
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="item-type">
        <xs:sequence>
            <xs:element name="item-code" type="xs:string"/>
            <xs:element name="quantity" type="xs:int"/>
        </xs:sequence>
    </xs:complexType>

</xs:schema>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<st:purchase-order xmlns:st="com.ebay.sales.types" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                   xsi:schemaLocation="com.ebay.sales.types file:///E:/webservices/namespace1.xsd">
    <st:item>
        <st:item-code>C54</st:item-code>
        <st:quantity>30</st:quantity>
    </st:item>
</st:purchase-order>
```

If in XSD elementFormDefault is qualified then in XML we have to prefix any short name

Default short name

```
<?xml version="1.0" encoding="UTF-8"?>
<purchase-order xmlns="com.ebay.sales.types"
                xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                xsi:schemaLocation="com.ebay.sales.types file:///E:/webservices/namespace1.xsd">
        <order-item>
                <item>
                        <item-code>C134</item-code>
                        <quantity>5423</quantity>
                </item>
        </order-item>
</purchase-order>
```

If we are giving short name as default type then no need to wrire any short name any where

➔We have to must ensure in XSD elementFormDefault must be qualified.

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

        <xs:element name="purchase-order" type="est:purchase-order-type"/>
        <xs:complexType name="purchase-order-type">
                <xs:sequence>
                        <xs:element name="order-item" type="est:order-item-type"/>
                </xs:sequence>
        </xs:complexType>
        <xs:complexType name="order-item-type">
                <xs:sequence>
                        <xs:element name="item" type="est:item-type"/>
                </xs:sequence>
        </xs:complexType>
        <xs:complexType name="item-type">
                <xs:sequence>
                        <xs:element name="item-code" type="xs:string"/>
                        <xs:element name="quantity" type="xs:int"/>
                </xs:sequence>
        </xs:complexType>

</xs:schema>
```

Import vs Include

If you want to refer one XSD to another XSD there are two options available

➔IMPORT

➔INCLUDE

INCLUDE

Include has to be used when the namespace of both the XSD documents are same.

IMPORT

Import has to be used when the namespace of both the XSD documents are different.

Q.If the namespace of two XSDs are same .why we are writing multiple XSD?

➔Let's say if we have a XSD document with more than 100 elements. Writing or managing in one single XSD is not so easy and is error prone (give) as well.

➔Instead we can split it in to two XSD's and can include one into another.

INCLUDE

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

    <xs:element name="shipping-address" type="shipping-address-type"/>
    <xs:complexType name="shipping-address-type">
        <xs:sequence>
            <xs:element name="address-line1" type="xs:string"/>
            <xs:element name="address-line2" type="xs:string"/>
            <xs:element name="city" type="xs:string"/>
            <xs:element name="state" type="xs:string"/>
            <xs:element name="zip" type="xs:string"/>
            <xs:element name="country" type="xs:string"/>
        </xs:sequence>
    </xs:complexType>
</xs:schema>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:est="com.ebay.sales.types"
        targetNamespace="com.ebay.sales.types" elementFormDefault="qualified">
    <xs:include schemaLocation="E:\webservices\address.xsd"/>
    <xs:element name="purchase-order" type="est:purchase-order-type"/>
    <xs:complexType name="purchase-order-type">
        <xs:sequence>
            <xs:element ref="est:shipping-address"/>
        </xs:sequence>
    </xs:complexType>
</xs:schema>
```

IMPORT

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

        <xs:element name="shipping-address" type="shipping-address-type"/>
        <xs:complexType name="shipping-address-type">
                <xs:sequence>
                        <xs:element name="address-line1" type="xs:string"/>
                        <xs:element name="address-line2" type="xs:string"/>
                        <xs:element name="city" type="xs:string"/>
                        <xs:element name="state" type="xs:string"/>
                        <xs:element name="zip" type="xs:string"/>
                        <xs:element name="country" type="xs:string"/>
                </xs:sequence>
        </xs:complexType>
</xs:schema>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
                xmlns:est="com.ebay.sales.types"
                xmlns:estd="com.ebay.sales.types.address"
                targetNamespace="com.ebay.sales.types"
                elementFormDefault="qualified" >
        <xs:import namespace="com.ebay.sales.types.address" schemaLocation="E:\webservices\address.xsd"/>
        <xs:element name="purchase-order" type="est:purchase-order-type"/>
        <xs:complexType name="purchase-order-type">
                <xs:sequence>
                        <xs:element ref="estd:shipping-address"/>
                </xs:sequence>
        </xs:complexType>
</xs:schema>
```

Important

We have to use <xs:import> in which one attribute is namespace="..." and
in side <xs:schema> we have declare one extra xmlns:...="..." which must be same as namespace

Working with any

➔Suppose we have a ticket counter with two bus travellers then we have to make these types of applications.

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

    <xs:element name="redbus" type="red:redbus-type"/>

    <xs:complexType name="redbus-type">
        <xs:sequence>
            <xs:element name="Company-name" type="xs:string"/>
            <xs:element name="branch" type="xs:string"/>
            <xs:element name="book" type="red:book-ticket-type"/>
        </xs:sequence>
    </xs:complexType>

    <xs:complexType name="book-ticket-type">
        <xs:sequence>
            <xs:any/>
        </xs:sequence>
    </xs:complexType>

</xs:schema>
```

If more no of vendors are available not recomended to import all the vendors and keep them in side <xs:choice> tag instead of this there is a easy way to use them in run time

```
<?xml version="1.0" encoding="UTF-8"?>
<red:redbus xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                        xsi:schemaLocation="http://redbus.com/account/types/address  file:///E:/webservices/ImportVsInclude/redbus.xsd
                                            http://sivani.com/travel/book   file:///E:/webservices/ImportVsInclude/sivani.xsd
                                            http://dj.com/travel/book file:///E:/webservices/ImportVsInclude/dj.xsd "
                        xmlns:red="http://redbus.com/account/types/address"
                        xmlns:sv="http://sivani.com/travel/book"
                        xmlns:dj="http://dj.com/travel/book" >

    <red:Company-name></red:Company-name>
    <red:branch></red:branch>
    <red:book>
        <dj:book>
            <dj:ticket-id>1220</dj:ticket-id>
            <dj:ticket-date>13/01/2015</dj:ticket-date>
            <dj:seat-no>24</dj:seat-no>
        </dj:book>
    </red:book>
</red:redbus>
```

OR

```
<sv:book>
    <sv:ticket-id>1220</sv:ticket-id>
    <sv:ticket-date>13/01/2015</sv:ticket-date>
    <sv:seat-no>24</sv:seat-no>
</sv:book>
```

In XML any ticket vendors implementation allowed if we are using <xs:any>

But we have to add explicitly vendor specific namespace and short name

Working with Nillable

➔How to use null in XML

```
<?xml version="1.0" encoding="utf-8"?>
<r2t:redbus-ticket  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.redbus.com/2016/ticket file:///E:/webservices/ImportVsInclude/Integration/redbusInt/redbus.xsd"
xmlns:r2t="http://www.redbus.com/2016/ticket">
        <r2t:redbus-ticket-id/>
        <r2t:member-id xsi:nil="true"/>
        <r2t:ticket-date>2016/11/11</r2t:ticket-date>
        <r2t:amount>1000</r2t:amount>
</r2t:redbus-ticket>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified" attributeFormDefault="qualified"
targetNamespace="http://www.redbus.com/2016/ticket" xmlns:r2t="http://www.redbus.com/2016/ticket">

        <xs:element name="redbus-ticket" type="r2t:redbus-ticket-type"/>
        <xs:complexType name="redbus-ticket-type">
                <xs:sequence>
                        <xs:element name="redbus-ticket-id" type="xs:string"/>
                        <xs:element name="member-id" type="xs:string" nillable="true"/>
                        <xs:element name="ticket-date" type="xs:string"/>
                        <xs:element name="amount" type="xs:float"/>
                </xs:sequence>
        </xs:complexType>
</xs:schema>
```

Difference between DTD and XSD

DTD	XSD
DTD stands for document type definition	XSD stands for XML Schema Documents
DTD's are not XML Type documents	XSD's are XML Type documents
DTD's are hard to learn as those are Not XML documents. So, an XML programmer has to know syntaxes in coding DTD	As XSD's are XML type documents it is easy for any programmer to work with XSD.
DTD's are not type safe, these will represents all the elements with data type as (#PCDATA).	XSD's are strictly typed, where the language has list of pre-defined data Types in it. While declaring the element you need tell whether this element is of What type?
DTD's don't allow you to create user defined Types.	XSD's allows you to create user defined data types using complexType declaration and using that type you can Create any number of elements.

JAX-P

JAX-P

➔JAX-P stands for Java API for XML Processing.
➔Initially in **Java there is no API that allows reading the contents of an XML document in an XML format,** Java programmers has to live with Java IO programming to read them in a text nature or need to build their own logic to read them in XML format.
➔Initially if you want to work with XML **then third party libraries** are you need to stick with vendor specific classes, as **there is no common API which facilities switching between them.**
➔So java has provided an API called JAX-P to read content of the XML

MAIN-FRAME Technology

➔ Earlier people used legacy application or some older application that is built on MAIN-FRAMES technology. It is the bigger machine than the any other machines in the earth having huge computation capacities & huge storage capacities.

➔ It is a client and server technology where the resources and the server is the one which is going to perform the operation & all the client s that are connected to the server are dummy clients. So **before the Distributed Technology the technology which is leading the market is called MAIN-FRAME Technology.** Which is also called Server Centric Application.
➔ When it comes to MAIN-FRAME it's a central server & clients architectures. Server and clients architectures where the entire process is taken care by the server itself.
➔ It is a highly secured system because the data and the processing of the data will be kept in one single place so no one can still.
➔ Modification or any changes in the data is very easy. Because all the data are placed in one single place. Upgrading of the system is very easy.
➔ But the disadvantage of this technology is , there is only one server that taking the load of any number of clients , if the server went down all the terminals are will become dummy terminals .
➔There is no way to make those terminals work. Because basically those terminal also not contains the processor as well. So the computers that are there in the client side are not at all the computer so the entire system will get crash when the central server has been failed.

➔ **So they wanted to replace the central server. So they cannot run the application in one CPU. So it is a machine critical application. Lakhs and millions of people are accessing this application. So high amount of performance is required. High amount of storage capacity is required. So we cannot run this application by place in normal system or normal CPU/Computer.**

DISTRIBUTED TECHNOLOGY

➔ **That is the reason distributed technology comes into picture**.
➔Breaks your application into multiple parts.
➔So run your application across multiple systems.
➔ As the system is not been centralized or not in one place as the system is distributed by multiple places.
➔So every system i.e. on which my application is running may not undergo crash so all the entire system will not be brought into huddle.
➔ As my application is running in multiple servers it will not going to crash and performance will get optimized. **So it is an alternate to a central server to distributed technology**

Where we can use JAX-P API

➔ In Web-Services provider is contains a class which contains the methods in which complex business logic is there. These methods will needs data i.e. is passed as an input in terms of object so it is built in java technology.
➔ So consumer wants to communicate with the provider. Consumer should have the data that is required as an input to the provider.
➔Consumer holds the data in terms of object because it is a java application.
➔But the same object of data cannot send over the network to the provider unless until we converts the object in to transferable format.
➔ Object cannot transferable over the network. Only bits & bytes are transferable over the network. Object has only memory representation it does not has bits representation. So object has to converts into bits & bytes.
➔ We have to converts this object into the form of bits and bytes which is transferable over the network. We can achieve this by using Serialization in java.
➔ But we should not use serialization to send the data over the network .

➔ **Serialization is something which is specific to java language only**. The non-java application that is there is on the other side cannot De-Serialized the data i.e. is been send in the form of bits and bytes by using the Serialization protocol.
➔ If the provider is a non-java application then it cannot de-serialized it. Interoperability is not there.
➔So to use Serialization protocol in exchanging the data between the systems that is two parties who want to exchange the data using the serialization must and should build in java technology. So serialization should not use to achieve Interoperability.

➔ Due to the above reason we should use Serialization representation format somehow some way we have to convert this object into special representation which can be converted into bits & bytes that can be transferable over the network.
➔ That's why converted this object into XML format. Consumer has to do it. Now transform this XML into bits & bytes. XML is character representation format that can be transformed in to bits & bytes. Send this data over the network.
➔Now provider receives the data in the form of bits & bytes. **Now converts this Bits & bytes into XML format & now convert this XML into object then this object can be set as input to my provider methods.** Now it performs the business logic. **Now it returns the object then converts this object into XML pass it to the consumer then consumer will receive the XML convert this XML into object then pass this object as input to consumer.**
➔ **Consumer converts the object into XML by using JAX-P. Provider reads the data From the XML by using JAX-P** and converts into an object. **So using JAX-P we can**

convert the object of data into XML & using JAX-P we can read the data from an XML converted into Object.

➔ **JAX-P is not meant for converting an XML into object. This is an API which helps us to reading and writing the data into an XML.**

How can we works with JAX-P?

➔ Initially there is no existing of XML in programming world. People used to store the data into character representation format using the text of characters into a text file.
➔ But later on due to the increasing demand of usage of data by the application people starting innovating & exploring something called XML.
➔ Earlier people used to store and share the data in the form of character representation using normal text file. The problem with this **is we cannot not validate the data whether the data is proper or improper.**
➔ **But if we store the data using XML there is a language like DTD or XSD which can be validate the data.** So the adoption of XML became huge in the market. XML not also touched every where it also reached to the java application as well & the need of using XML as part of the java application is came into the picture.
➔But initially the java has not provided any API to work with XML. So the java programmer increasing demand badly to provide support for XML by providing API. But java has not showing any interest to providing support for XML because java is open source. So java application has not any way to read the content of an XML. SO this problem is identified by the software vendors who are present inside the market
➔ So there is a gap between XML and java and huge amount of people are looking for filling the gaps by provide some API to easily program the XML in java. That's why Third Party Vendor in the market comes into the picture.
➔ At that time there is no API called JAX-P. Programmer has to read the content of an XML as Strings and tokenize them as per the tags using IO-Streams. So for this programmer has to write the complex logic because there is no JAX-P API.
➔ So Third-Party Vendor says to java programmers instead of you writing the complex logics we will provide classes. This classes contains the complex logic for reading the XML instead of using IO-Stream.
➔ After many software vendors started developing their own libraries, sun has finally Released JAX-P API which allows us to work with XML documents.
➔ As indicated
JAX-P is an API which means not complete in nature, so we need implementations Of JAX-P API and there are many implementations of JAX-P API available for e.g..

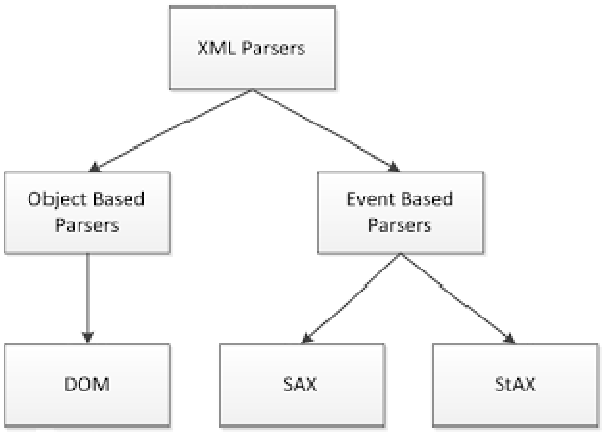
- **➔Crimson**
- **➔ Xerces2 (default)**
- **➔Oracle V2 Parser etc…**

➔ Xerces2 is the default parser that would be shipped as part of JDK1.5 onwards. This Indicates if you are working on JDK1.5 onwards you don't need any separate Jar's to work With JAX-P API.

XML Processing Methodologies

➔ There are multiple methods of reading the XML documents exists, those are SAX and DOM.

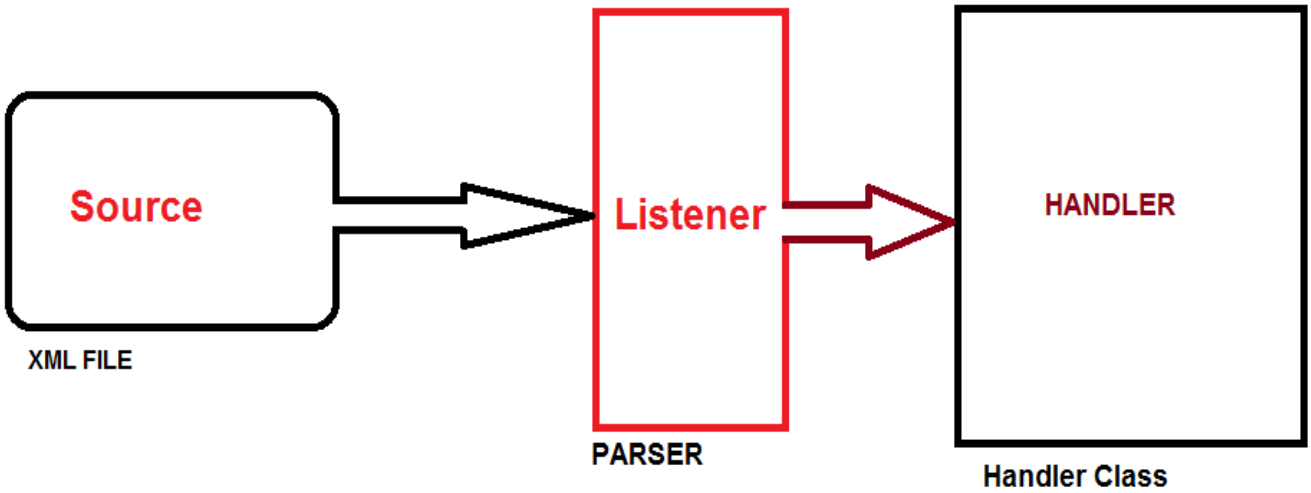
➔ SAX and DOM are the universal methodologies of processing (reading) XML Documents irrespective of a particular technology.



SAX (Simple Access for XML)

➔ SAX stands for Simple Access for XML it is a methodology that allows reading XML documents in an event based processing model. Any event based Processing model contains three actors like.

- ➔ **Source** **(XML FILE)**
- ➔ **Listener** **(PARSER)**
- ➔ **Event Handler** **(HANDLER)**



SOURCE

Source is the originator of event, who can raise several types of events. Events could be of multiple types if we consider AWT, we can consider source as a button, which is capable of raising several types of events like button click, mouse move, key pressed etc.

LISTENER

Listener is the person who will listens for an event from the source. Listener listens for a specific type of event from the source and **triggers an even handling method on the Event handler to process it.**

EVENT HANDLER

Once a listener captures an event, in order to process it, it calls a method on the event handler class. As there are different types of events, to handle them the event handler contains several methods each target to handle and process a specific type of event.

➔ SAX is a sequential processing model, which process XML elements sequentially from top to the bottom.
➔ While processing the document, it places a pointer to the document and increments sequentially from the top.
➔ Based on the type of element it is pointing to and it raises a respective event, which will notify the listener to handle and process it.

➔ So here source is the XML document which can raise several types of events based on the type of element it is pointing to…

➔startDocument
➔startElement
➔characters
➔endElement
➔endDocument

➔ Listener is the parser who will reads the XML document and triggers a method call on the Handler.
➔ Event handler contains several methods to handle various types of events.
➔ SAX is very fast in processing XML documents when compared with DOM and consumes less amount of memory
➔ As at any given point of time it loads only one element into the memory and process.
➔Using SAX we can only read the XML document, we cannot modify/create a document

➔JAX-P is an API which contains interfaces/abstract classes and few concrete implementations.
➔ SAXParser is also an interface, in order to instantiate it, we need to find the respective implementation of it.
➔ SAXParserFactory which would be able to Instantiate the implementation of SAXParser.

➔**SAXParserFactory is designed based on AbstractFactory Design Pattern.**

Abstract Factory Design Pattern

```
public class Test {
	public static void main(String[] args) {
		AbstractFactoryJXmlReaderFactory factory=AbstractFactoryJXmlReaderFactory.createReader();
		IJXmlReader reader=factory.getJIXmlReader();
		reader.JXmlRead();
	}
}
```

```
public abstract class AbstractFactoryJXmlReaderFactory {
	public abstract IJXmlReader getJIXmlReader();
	public  static AbstractFactoryJXmlReaderFactory createReader(){
		String className=System.getProperty("className");
		AbstractFactoryJXmlReaderFactory factory = null;
		try {
			factory=(AbstractFactoryJXmlReaderFactory) Class.forName(className).newInstance();
		} catch (InstantiationException | IllegalAccessException | ClassNotFoundException e) {
			e.printStackTrace();
		}
		return factory;
	}
}
```

```
public class ApacheJXmlReaderFactory  extends AbstractFactoryJXmlReaderFactory{
	private static IJXmlReader reader;
	@Override
	public IJXmlReader getJIXmlReader() {
		if (reader==null) {
			reader=new ApacheJXmlReader();
		}
		return reader;
	}
}
```

```
public class SunJXmlReaderFactory extends AbstractFactoryJXmlReaderFactory  {
	private static IJXmlReader reader;
	@Override
	public IJXmlReader getJIXmlReader() {
		if (reader==null) {
			reader=new SunJXmlReader();
		}
		return reader;
	}
}
```

```
public interface  IJXmlReader {
	public void JXmlRead();
}
```

```
public class ApacheJXmlReader implements IJXmlReader  {
	@Override
	public void JXmlRead() {
		IJXmlElement element=new ApacheJXmlElementImpl();
		String tag=element.getTag();
		String value=element.getValue();
		System.out.println("Apache Tag Name :"+tag+"\t Apache Value:"+value);
	}
}
```

```
public class SunJXmlReader implements IJXmlReader{
	public void JXmlRead() {
		IJXmlElement element=new SunJXmlElementImpl();
		String tag=element.getTag();
		String value=element.getValue();
		System.out.println("Sun Tag Name :"+tag+"\t SunValue:"+value);
	}
}
```

```
public interface IJXmlElement {
	String  getTag();
	String getValue();
}
```

```
public class ApacheJXmlElementImpl implements IJXmlElement{
	@Override
	public String getTag() {
		return "TagName : class";
	}
	@Override
	public String getValue() {
		return "Tag value : A";
	}
}
```

```
public class SunJXmlElementImpl implements IJXmlElement  {
	@Override
	public String getTag() {
		return "TagName : class";
	}
	@Override
	public String getValue() {
		return "Tag value : A";
	}
}
```

-DclassName= ApacheJXmlReaderFactory

➔To instantiate SAXParser.

```
SAXParserFactory saxParserFactory = SAXParserFactory.newInstance();
SAXParser parser = saxParserFactory.newSAXParser();
```

➔Along with the parser, we need a Handler class to handle the event and process it. A handler class would be written by extending it from DefaultHandler, we can override the methods to process respective events. Below is the example showing a class extending from DefaultHandler.

```
public class MyHandler extends DefaultHandler{
	@Override
	public void startDocument() throws SAXException {
		System.out.println("Start Reading the XML");
	}
	@Override
	public void startElement(String uri, String localName, String qName, Attributes attributes) throws SAXException {
		System.out.print("<"+qName+">");
	}
	@Override
	public void characters(char[] ch, int start, int length) throws SAXException {
		System.out.print(new String(ch,start,length));
	}
	@Override
	public void endElement(String uri, String localName, String qName) throws SAXException {
		System.out.println("</"+qName+">");
	}
	@Override
	public void endDocument() throws SAXException {
		System.out.println("End Reading the XML");
	}
}
```

➔In the above class we are overriding five methods which would be triggered based on the type of elements the parser is pointing to on the source XML document.

- ⇨ **startDocument()** – would be triggered at the start of the XML document
- ⇨ **startElement()** – Whenever the parser encounters the starting element, it raises this method call by passing the entire element information to the method.
- ⇨ **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.
- ⇨ **endElement()**-would be triggered by the parser, when it encounters a closing element or ending element.
- ⇨ **endDocument**-would be triggered by the parser once it reaches end of the document.

➔Once the handler has been created, we need to call the parser class parse() method by passing the source as XML and the handler as the object of handler class which we created just now. So, that the parser will reads the XML elements and triggers a respective method call on the handler object provided.

The below program will reads the XML document sequentially and prints it on to The console.

```
public class Test {
	public static void main(String[] args) throws ParserConfigurationException, SAXException, IOException {
		SAXParserFactory factory=SAXParserFactory.newInstance();
		SAXParser parser=factory.newSAXParser();
		parser.parse(new File("D:/workspaceWS/SAX/resources/irctc.xml"), new MyHandler());
	}
}
```

Get the Quantity from Purchase Order

```
<?xml version="1.0" encoding="UTF-8"?>
<to:purchase-order xmlns:to="http://www.travel.org/trip" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                   xsi:schemaLocation="http://www.travel.org/trip file:///E:/webservices/po.xsd">
      <to:order-item>
            <to:item>
                  <to:item-code>c121</to:item-code>
                  <to:quantity>34</to:quantity>
            </to:item>
            <to:item>
                  <to:item-code>H028</to:item-code>
                  <to:quantity>45</to:quantity>
            </to:item>
      </to:order-item>
      <to:shipping-address>
            <to:address-line1>544</to:address-line1>
            <to:address-line2>mytrivanam</to:address-line2>
            <to:city>Hyderabad</to:city>
            <to:state>ts</to:state>
            <to:zip>700058</to:zip>
            <to:country>india</to:country>
      </to:shipping-address>
</to:purchase-order>
```

```
public class MyHandler extends DefaultHandler{
      String elementName;
      int totalQty;
      @Override
      public void startDocument() throws SAXException {
            System.out.println("Start Reading the XML");
      }
      @Override
      public void startElement(String uri, String localName, String qName, Attributes attributes) throws SAXException {
            System.out.print("<"+qName+">");
            if (qName.equals("to:quantity")) {
                  this.elementName=qName;
            }
      }
      @Override
      public void characters(char[] ch, int start, int length) throws SAXException {
            String value=new String(ch,start,length);
            System.out.print(value);
            if (elementName!=null && elementName.length()>0) {
                  if (elementName.equals("to:quantity")) {
                        totalQty +=Integer.parseInt(value);
                  }
            }
            elementName=null;          Make it as null otherwise it will not work
      }
      @Override
      public void endElement(String uri, String localName, String qName) throws SAXException {
            System.out.println("</"+qName+">");
      }
      @Override
      public void endDocument() throws SAXException {
            System.out.println("End Reading the XML");
      }
      public void getQty(){
            System.out.println("Total Item Quantity:"+totalQty);
      }
}
```

```
public class Test {
      public static void main(String[] args) throws ParserConfigurationException, SAXException, IOException {
            MyHandler handler=new MyHandler();

            SAXParserFactory factory=SAXParserFactory.newInstance();
            SAXParser parser=factory.newSAXParser();
            parser.parse(new File("D:/workspaceWS/SAX/resources/po.xml"),handler );
            handler.getQty();
      }
}
```

DOM (Document Object Model)

➔DOM stands for Document Object Model. Dom is a universal access methodology that is used for reading the content of an XML.

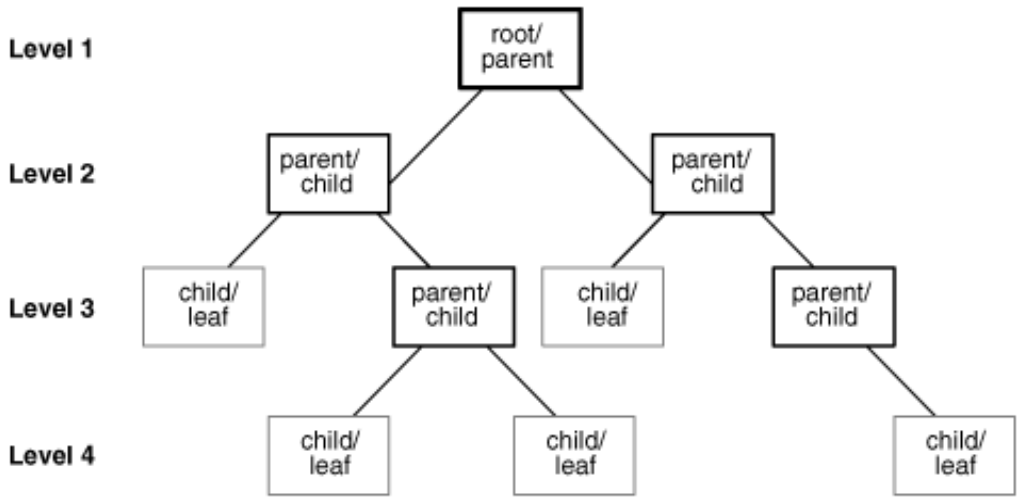
➔It is not specific to a language rather it is adopted by all the language in the market.

➔SAX will permit us to read the content of an XML sequentially but **DOM permits us to read the content of an XML in a Hierarchical model.**

➔ DOM Engine will read the complete content of the XML from top to bottom.
➔It just loads the complete content of the XML into JVM memory and constructs the hierarchical representation of that XML by representing in the form of Tree structure (as parent and child).
➔Every thing that is there part of the XML will become a node. These nodes are related with each other based on the relationship they have within the XML.
➔These nodes are multiple types.

- ⇨ Elements Node
- ⇨ Data Nodes
- ⇨ Attribute Nodes.
- ⇨ Document Node
- ⇨ Text Node
- ⇨ Empty node

➔ As it builds a logical tree structure in the JVM memory, in DOM we can navigate between various elements in random order rather than sequential.
➔Relationships between these nodes will be established in terms of parent and child. It means forward I can move, backward also I can move by asking the pointer to get me the next one or get me the previous one.
➔To navigate between various elements in DOM, it has provided convenient methods by calling which we can traverse between them.



➔As explained above except the root element all the other elements have a parent node. Children of same parent are called brother/sister in other words siblings.

➔JAX-P provides a way to work with these nodes in Java, JAX-P has provided methods like **getFirstChild(), getChildNodes** etc to access children of a parent. In the same way to access the siblings it has provided methods like **getNextSiblings().**
➔DOM has provided many mechanisms like get Element by Tag name. So we don't need to traverse between all the elements rather we can randomly access a specific node within the tree. That's why DOM provided to navigational API.

➔Traversal API
➔Random Access API

How to read the content of an XML using DOM Methodology?

➔ We have to read the content of an XML by using DOM methodology in a Hierarchical model representing the entire content of the XML and create a Tree Structure carrying the content of the XML as nodes of the Tree in the JVM memory.
➔ Document is a class that holds the complete structure of the XML as a Tree representation been constructed and placed inside the JVM memory. If we have one Document means it represents one XML document as a Tree representation inside JVM memory.
➔ Document is class it comes from JAX-P API. API's are always partial. **So document is may be an Interface or Abstract class**. So we don't know how to create the object of Document. **So we have to go to the Document factory**. But it is always not same. So there is class called **Document Builder.**
➔**Document Builder is a class again that is coming from JAX-P API**. So it also may be an Interface or Abstract class. So it is not a concrete class. So I don't know how to create the object of DocumentBuilder. **So I have to go to the DocumentBuilderFactory**. Again it is also come from JAX-P API. I don't know how to create the object of DocumentBuilder Factory. So there is a static factory method that creates the object of DocumentBuilder Factory.
➔ DocumentBuilderFactory ➔DocumentBuilder➔Document.

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

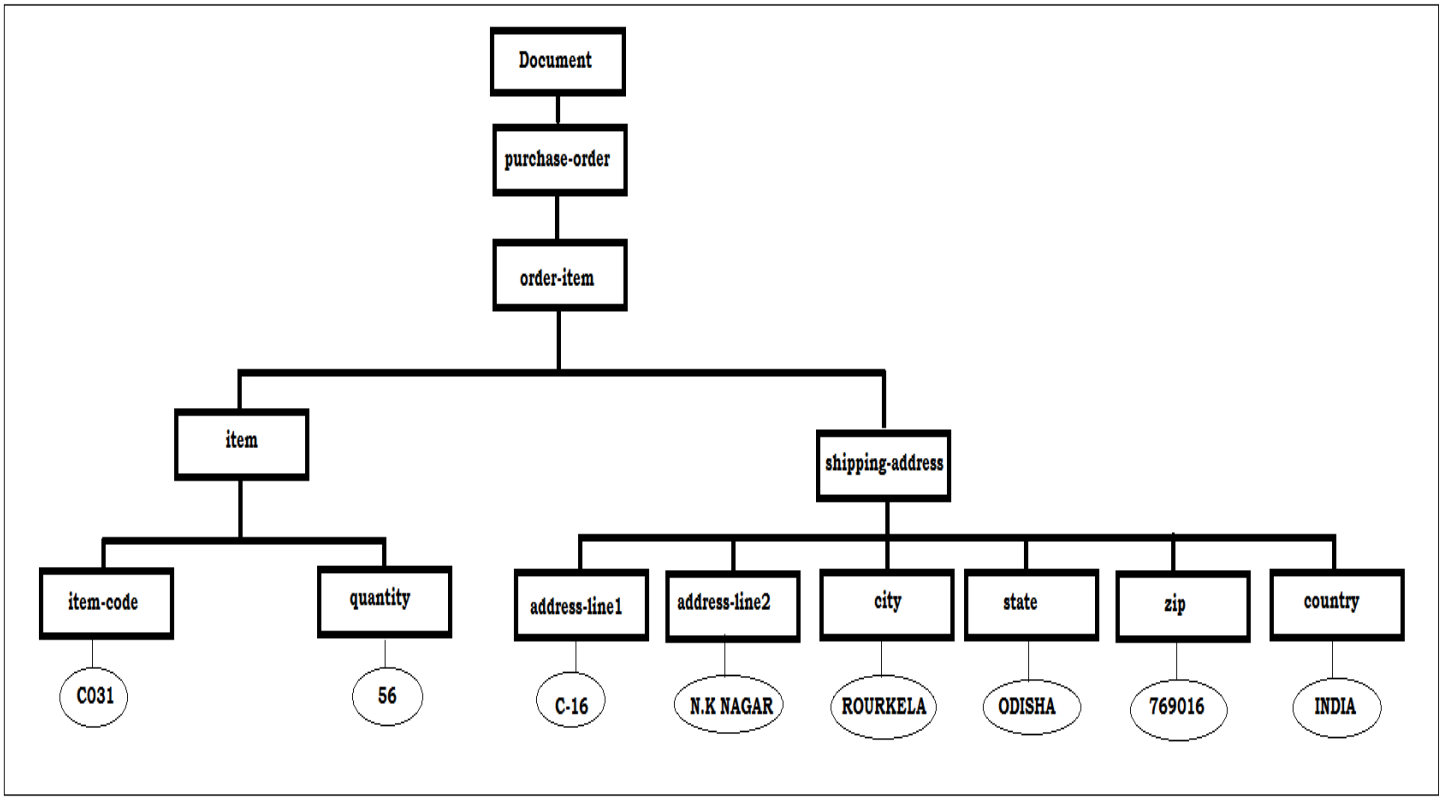
➔ After creating the Document object, it points to the top of the Document which is called Document Node. Now we need to use methods to navigate between the elements of it.

➔ Below sample program shows how to navigate to an element order Items in the earlier XML.

XML

```
<?xml version="1.0" encoding="UTF-8"?>
<purchase-order xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                xmlns="http://www.ebay.com/sales/types"
                xsi:schemaLocation="http://www.ebay.com/sales/types file:///E:/webservices/po.xsd">

    <order-item>
        <item>
            <item-code>C031</item-code>
            <quantity>56</quantity>
        </item>
    </order-item>
    <shipping-address>
        <address-line1>C-16</address-line1>
        <address-line2>N.K NAGAR</address-line2>
        <city>ROURKELA</city>
        <state>ODISHA</state>
        <zip>769016</zip>
        <country>INDIA</country>
    </shipping-address>
</purchase-order>
```

Hierarchical Model



Random Access

```
public class DomAccessor {
    private static Document doc=null;
    private final static String  XML_FILE="D:/workspaceWS/DOM/src/po.xml";
    public static void main(String[] args) throws SAXException {
        try {
            DocumentBuilderFactory factory=DocumentBuilderFactory.newInstance();
            DocumentBuilder builder=factory.newDocumentBuilder();
            doc=builder.parse(new File(XML_FILE));
            System.out.println(doc.getFirstChild().getNodeName());
            System.out.println(doc.getFirstChild().getFirstChild().getNextSibling().getNodeName());
            System.out.println(doc.getFirstChild().getFirstChild().getNextSibling().getFirstChild().getNextSibling().getNodeName());
            System.out.println(doc.getFirstChild().getFirstChild().getNextSibling().getFirstChild().getNextSibling().getFirstChild().getNextSibling().getNodeName());
        } catch (ParserConfigurationException | SAXException | IOException e) {
            e.printStackTrace();
        }
    }
}
```

➔ Instead to print the entire XML document on to the console we need to traverse all the elements recursively. As you know DOM loads the XML document represents it in a Hierarchical manner (Tree structure), any Hierarchical representations needs recursion technique as the things would be in parent and child nested format.
➔If we take Binary tree, the tree traversal algorithms are in order, pre-order and post-order which are recursive techniques based traversal algorithms.

Traversal Access

```
public static void  traverse(Node node){
    if (node!=null) {
        switch(node.getNodeType()){
            case Node.DOCUMENT_NODE:
                System.out.println("<? xml version='1.0' encoding='UTF-8'>");
                traverse(node.getFirstChild());
                break;
            case Node.ELEMENT_NODE:
                System.out.print("<"+node.getNodeName()+">");
                    NodeList children=node.getChildNodes();
                    if (children!=null) {
                        for (int i = 0; i < children.getLength(); i++) {
                            Node child=children.item(i);
                            traverse(child);
                        }
                    }
                    System.out.println("</"+node.getNodeName()+">");
                    break;
            case Node.TEXT_NODE:
                System.out.print(node.getNodeValue());
                break;
        }
    }
}
```

Example: to read total content of XML using DOM Recursive Technique.

```
public class DomAccessor {
	private static Document doc=null;
	private final static String  XML_FILE="D:/workspaceWS/DOM/src/po.xml";

	public static void  traverse(Node node){
		if (node!=null) {
			switch(node.getNodeType()){
				case Node.DOCUMENT_NODE:
					System.out.println("<? xml version='1.0' encoding='UTF-8'>");
					traverse(node.getFirstChild());
					break;
				case Node.ELEMENT_NODE:
					System.out.print("<"+node.getNodeName()+">");
						NodeList children=node.getChildNodes();
						if (children!=null) {
							for (int i = 0; i < children.getLength(); i++) {
								Node child=children.item(i);
								traverse(child);
							}
						}
						System.out.println("</"+node.getNodeName()+">");
						break;
				case Node.TEXT_NODE:
					System.out.print(node.getNodeValue());
					break;
			}
		}
	}

	public static void main(String[] args) throws SAXException {
		try {
			DocumentBuilderFactory factory=DocumentBuilderFactory.newInstance();
			DocumentBuilder builder=factory.newDocumentBuilder();
			doc=builder.parse(new File(XML_FILE));
			traverse(doc);
		} catch (ParserConfigurationException | SAXException | IOException e) {
			e.printStackTrace();
		}
	}
}
```

Validating XML with an XSD using JAX-P.

➔ **While parsing an XML the parser only checks for well-formness of an XML it never checks whether it is valid or invalid.** But in a typical business transaction we should first validate the XML before parsing it.
➔ To validate an XML we need either a DTD/XSD.
➔ To validate an XML first of all we need to load the contents of XSD and keep in a java object called Schema.
➔Schema is coming from JAX-P API which means it is an abstract class. The implementation for the abstract class Schema will be there with default implementation (Xerces2).
➔ **As we don't know who is the implementation for Schema class we need one more factory class called Schema Factory. While creating the SchemaFactory we need to pass the input as W3C XML Schema Namespace URI as input to indicate we are going to read the content of XML Schema compliant XSD.**
➔ Once we create the **SchemaFactory** we can call **newSchema()** method on it by passing file pointing to XSD as input.
➔ After creating the schema object call a method **newValidator()** will gives you the **Validator object**. On this object you can call the method called validate by passing as input.

➔**If the XML passed is invalid, it throws Exception otherwise will not return anything.**

DOM VALIDATION.

```
public class DomAccessor {
	private static Document doc=null;
	private final static String  XSD_PATH="D:/workspaceWS/DOMValidate/src/po.xsd";
	private final static String  XML_PATH="D:/workspaceWS/DOMValidate/src/po.xml";
	public static void main(String[] args){
		try {

			SchemaFactory sfactory=SchemaFactory.newInstance(XMLConstants.W3C_XML_SCHEMA_NS_URI);
			Schema schema=sfactory.newSchema(new File(XSD_PATH));
			Validator validator=schema.newValidator();
			validator.validate(new StreamSource(new File(XML_PATH)));

			DocumentBuilder builder=DocumentBuilderFactory.newInstance().newDocumentBuilder();
			doc=builder.parse(new File(XML_PATH));
			System.out.println(doc.getFirstChild().getNodeName());


		} catch (ParserConfigurationException | SAXException | IOException e) {
			e.printStackTrace();
		}
	}
}
```

SAX VALIDATION.

```
public class Test {
	private final static String  XSD_PATH="D:/workspaceWS/DOMValidate/src/po.xsd";
	private final static String  XML_PATH="D:/workspaceWS/DOMValidate/src/po.xml";
	public static void main(String[] args) throws ParserConfigurationException, SAXException, IOException {
		MyHandler handler=new MyHandler();
		SchemaFactory sfactory=SchemaFactory.newInstance(XMLConstants.W3C_XML_SCHEMA_NS_URI);

		Schema schema=sfactory.newSchema(new File(XSD_PATH));
		Validator validator=schema.newValidator();
		validator.validate(new StreamSource(new File(XML_PATH)));

		SAXParserFactory factory=SAXParserFactory.newInstance();
		SAXParser parser=factory.newSAXParser();
		parser.parse(new File(XML_PATH),handler );
		handler.getQty();
	}
}
```

Difference between SAX & DOM

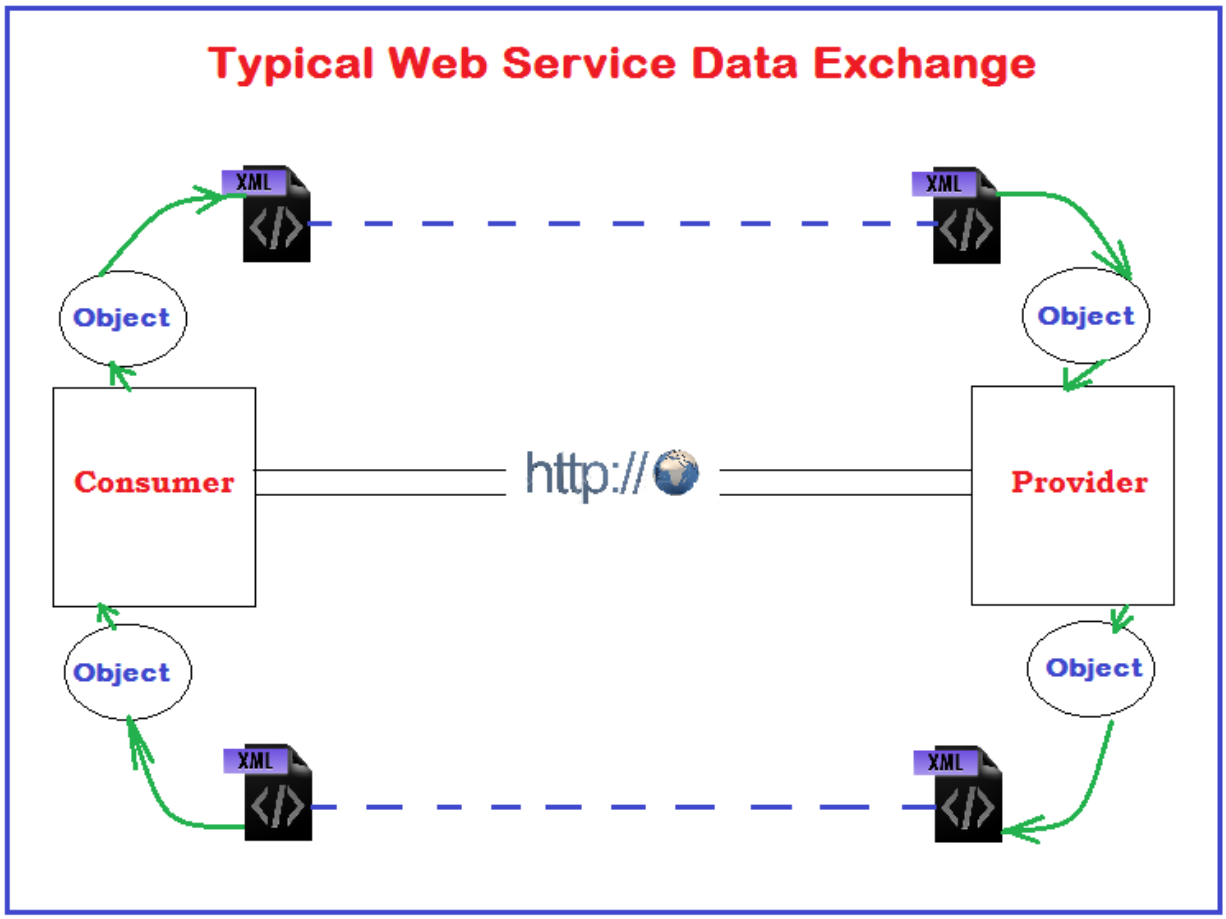
SAX	DOM
➔SAX stands for Simple Access for XML API.	➔DOM Stands for Document Object Model.
➔It is an event based processing model.	➔It is a hierarchical processing model where it loads the entire XML In memory and represents in tree structure.
➔It Process the XML sequentially one after the other from top to bottom.	➔Dom supports random access of element.
➔It consumes less amount of memory as at any point of time load only one element into the memory.	➔It consumes huge amount of memory as it loads the entire XML and forms tree structure (If XML Document is 1 kb it takes 1 kb **X** 10 kb memory hold in memory.)
➔It is faster when compared with DOM as it doesn't need to load or form anything.	➔It is slow when compare with SAX as it has to represent the XML by loading it.
➔SAX is a readonly API, which means we can just read the XML document, but we cannot modify or create new.	➔DOM is a read and write API using which we can read/modify/create an XML Document.

➔In order to overcome the both the side dis-advantages in **SAX** and **DOM** new processing model came which is **STAX**
➔**STAX** stands for Streaming API for XML.
➔This is unlike SAX event based or DOM hierarchical based, it is a pull based event parser.

JAX-B

JAX-B (design time tool)

➔ JAX-B stands for Java Architecture for XML Binding. It is the API that is used for Binding Java Object to XML document and XML documents to Java Object back.
➔ We can read the content of the XML using JAX-P API. But JAX-P allows us to read the content of an XML in terms of characters of data but if our application programmers want the data to be access in the terms of Objects of data then we should go for JAX-B API.
➔ If the consumer wants to send some information to the provider rather than converting it into serialized stream of data.
➔Programmer has to convert the **Java Object data to XML representation** and should send to Provider. After provided received the **XML he has to convert it to Java Object** and process it. Again Provider instead of returning Java Object as response, Provider should convert the response **Object to XML** and should send it to consumer as response. The consumer has to convert that incoming **XML data to Object** to use it.



➔We should not use Object serialization for exchanging data in web services rather we should convert them to XML.
➔In-order to work with XML java has already provided API's for dealing with XML data which is JAX-P.
➔ In JAX-P we can use either SAX or DOM parser to convert Java Object to XML and XML to Java Object.

➔ Even though we can achieve this with JAX-P, a programmer has to write manually the parser's dealing with this, and more over there are limitations with JAX-P.
➔ Initially Java doesn't have XML Binding supported API's.
➔ There are lot of third party XML binding libraries are released by software vendors in Java like

- ⇨ **JibX**
- ⇨ **Castor**
- ⇨ **XML Beans**
- ⇨ **Apache Data binding API**
- ⇨ **Javolution etc...**

➔ Then SUN has realized and released its own API for dealing with XML Binding in Java "JAX-B API". And even released implementations for it called JAX-B RI (Reference Implementation).
➔ Along with that there is an open source implementations for JAX-B API is JaxMe provided by Apache.
➔ Basically XML Binding API's are categorized into two types of tools.

- ⇨ Design Time tools
- ⇨ Runtime tools

Design Time tools

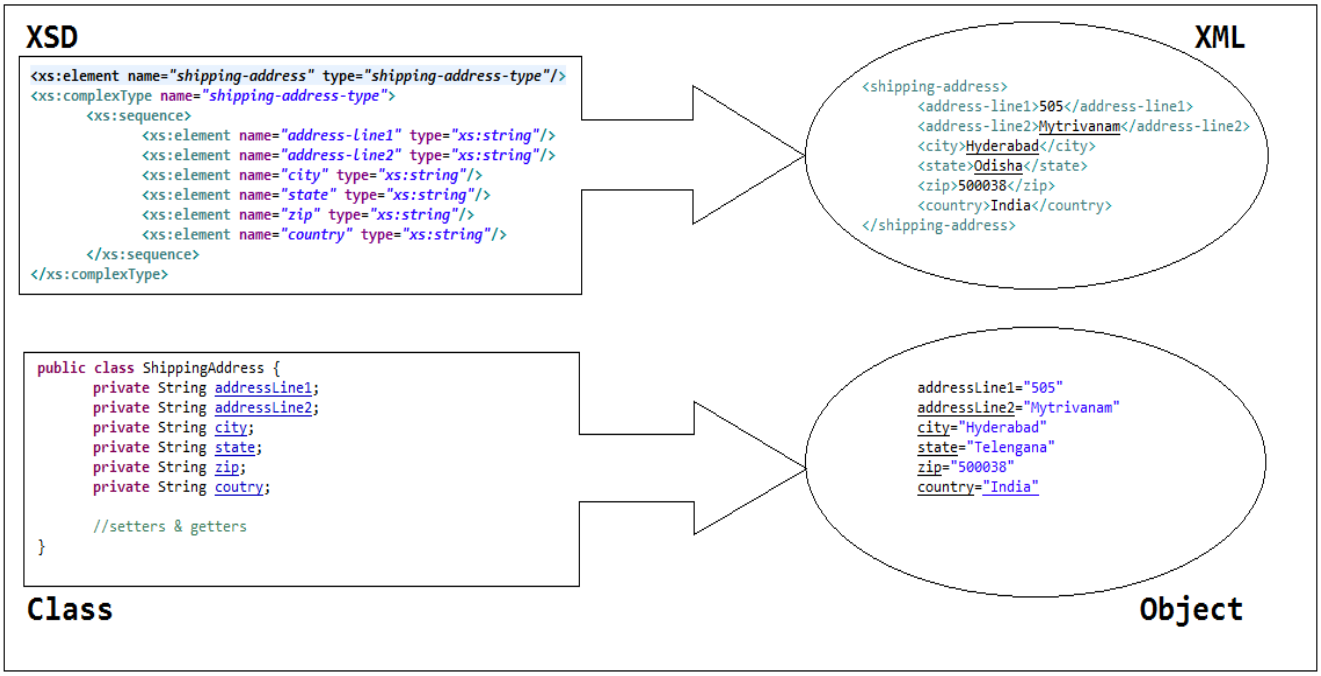
In these types of tools, there will be a Binding compiler which will take a Schema or DTD as input and generated supported classes for converting the XML to Object and Object to XML

Runtime tools

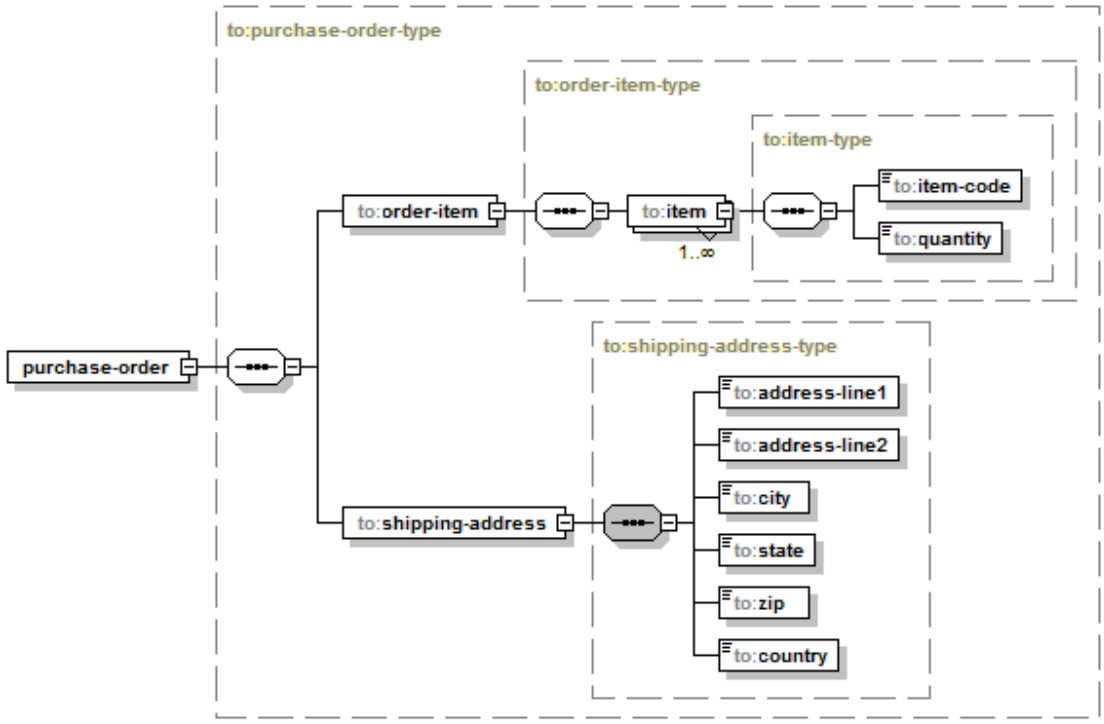
These will not generate any classes rather on runtime will take the XML as an input and based the elements names, matches them with corresponding attribute names of the class and will convert XML into Object of the class.
➔When it comes to JAX-B API it is a design time tool,
➔Where it will come up with its own compiler to generated classes to support the binding process.

Architecture

➔XML document is structure of an XSD document so XML is called as Instance of that XSD.
➔In Java, Object structure would be defined by a class.
➔But In XML, structure of an XML Is defined by an XSD document.



➔Classes and XSD documents both are representing the structure.
➔The XML document representing the structure of that XSD can be converted into its associated Java class Objects.
➔Even the vice versa would also be possible.
➔A ComplexType in XSD is equal to a Java class in Java.
➔To represent the structure of an XSD document in Java, it is nothing but composition of various Java classes representing its structure.




```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:to="http://www.ebay.com/sales/types"
           targetNamespace="http://www.ebay.com/sales/types" elementFormDefault="qualified" attributeFormDefault="qualified">
    <xs:element name="purchase-order" type="to:purchase-order-type"/>
    <xs:complexType name="purchase-order-type">
        <xs:sequence>
            <xs:element name="order-item" type="to:order-item-type"/>
            <xs:element name="shipping-address" type="to:shipping-address-type"/>
        </xs:sequence>
    </xs:complexType>                                                                    4
    <xs:complexType name="shipping-address-type">
        <xs:sequence>
            <xs:element name="address-line1" type="xs:string"/>
            <xs:element name="address-line2" type="xs:string"/>
            <xs:element name="city" type="xs:string"/>
            <xs:element name="state" type="xs:string"/>
            <xs:element name="zip" type="xs:string"/>
            <xs:element name="country" type="xs:string"/>
        </xs:sequence>
    </xs:complexType>                                                                    3
    <xs:complexType name="order-item-type">
        <xs:sequence>
            <xs:element name="item" type="to:item-type" minOccurs="1" maxOccurs="unbounded"/>
        </xs:sequence>
    </xs:complexType>                                                                    2
    <xs:complexType name="item-type">
        <xs:sequence>
            <xs:element name="item-code" type="xs:string"/>
            <xs:element name="quantity" type="xs:int"/>
        </xs:sequence>
    </xs:complexType>                                                                    1
</xs:schema>
```

➔In the above XSD we have one element and four ComplexType declarations, as we discussed earlier a ComplexType declarations in XSD is equal to a Java class in Java. So from the above we can derive four Java classes.

item-type ComplexType

ComplexType

```
<xs:complexType name="item-type">
        <xs:sequence>
                <xs:element name="item-code" type="xs:string"/>
                <xs:element name="quantity" type="xs:int"/>
        </xs:sequence>
</xs:complexType>
```

Class

```
package com.ebay.sales.types;


@XmlAccessorType(AccessType.FIELD)
@XmlType(name = "item-type", propOrder = {"itemCode","quantity"})
public class ItemType {

   @XmlElement(name = "item-code", namespace = "http://www.ebay.com/sales/types")
    protected String itemCode;

    @XmlElement(namespace = "http://www.ebay.com/sales/types", type = Integer.class)
    protected int quantity;

   //getters & Setters
}
```

order-item-type ComplexType

ComplexType

```
<xs:complexType name="order-item-type">
        <xs:sequence>
                <xs:element name="item" type="to:item-type" minOccurs="1" maxOccurs="unbounded"/>
        </xs:sequence>
</xs:complexType>
```

Class

```
package com.ebay.sales.types;

@XmlAccessorType(AccessType.FIELD)
@XmlType(name = "order-item-type", propOrder = {"item"})
public class OrderItemType {

    @XmlElement(namespace = "http://www.ebay.com/sales/types")
    protected List<ItemType> item;

    public List<ItemType> getItem() {
        if (item == null) {
            item = new ArrayList<ItemType>();
        }
        return this.item;
    }
}
```

shipping-address-type ComplexType

ComplexType

```
<xs:complexType name="shipping-address-type">
    <xs:sequence>
        <xs:element name="address-line1" type="xs:string"/>
        <xs:element name="address-line2" type="xs:string"/>
        <xs:element name="city" type="xs:string"/>
        <xs:element name="state" type="xs:string"/>
        <xs:element name="zip" type="xs:string"/>
        <xs:element name="country" type="xs:string"/>
    </xs:sequence>
</xs:complexType>
```

Class

```
package com.ebay.sales.types;
@XmlAccessorType(AccessType.FIELD)
@XmlType(name = "shipping-address-type", propOrder = {"addressLine1","addressLine2","city","state","zip","country"})
public class ShippingAddressType {

   @XmlElement(name = "address-line1", namespace = "http://www.ebay.com/sales/types")
   protected String addressLine1;
   @XmlElement(name = "address-line2", namespace = "http://www.ebay.com/sales/types")
   protected String addressLine2;
   @XmlElement(namespace = "http://www.ebay.com/sales/types")
   protected String city;
   @XmlElement(namespace = "http://www.ebay.com/sales/types")
   protected String state;
   @XmlElement(namespace = "http://www.ebay.com/sales/types")
   protected String zip;
   @XmlElement(namespace = "http://www.ebay.com/sales/types")
   protected String country;

   //getters & setters
}
```

purchase-order-type complexType

complexType

```
<xs:complexType name="purchase-order-type">
    <xs:sequence>
        <xs:element name="order-item" type="to:order-item-type"/>
        <xs:element name="shipping-address" type="to:shipping-address-type"/>
    </xs:sequence>
</xs:complexType>
```

Class

```
package com.ebay.sales.types;

@XmlAccessorType(AccessType.FIELD)
@XmlType(name = "purchase-order-type", propOrder = {"orderItem","shippingAddress"})

@XmlRootElement
public class PurchaseOrderType {

   @XmlElement(name = "order-item", namespace = "http://www.ebay.com/sales/types")
   protected OrderItemType orderItem;
   @XmlElement(name = "shipping-address", namespace = "http://www.ebay.com/sales/types")
   protected ShippingAddressType shippingAddress;

   //getters & setters
}
```

it indicates in FIELD level we are using annotations

Class

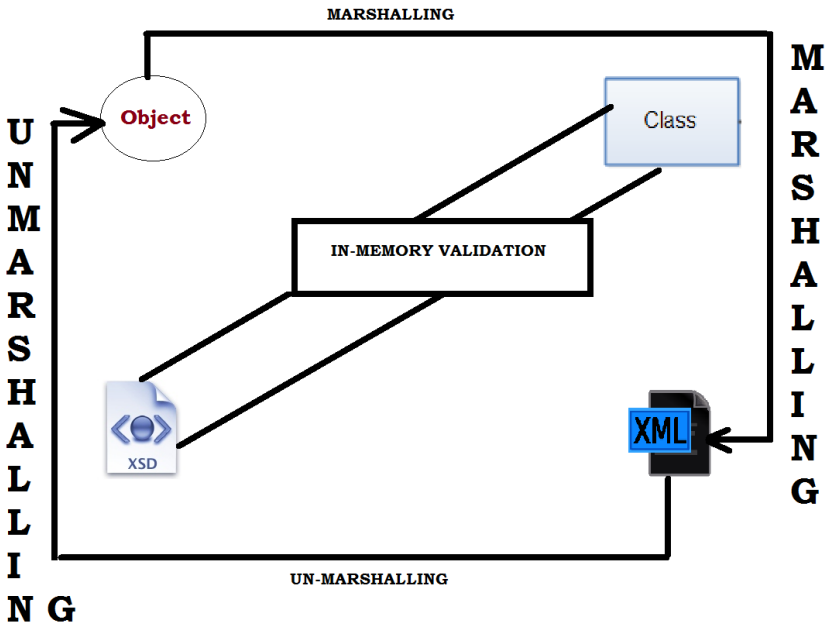
```
package com.ebay.sales.types;

@XmlAccessorType(AccessType.FIELD)
@XmlType(name = "purchase-order-type", propOrder = {"orderItem","shippingAddress"})

@XmlRootElement
public class PurchaseOrderType {

   @XmlElement(name = "order-item", namespace = "http://www.ebay.com/sales/types")
   protected OrderItemType orderItem;
   @XmlElement(name = "shipping-address", namespace = "http://www.ebay.com/sales/types")
   protected ShippingAddressType shippingAddress;

   //getters & setters
}
```

Root Element Name

it indicates sequence of elements in XSD

Tracking between element name from which namespace and attribute name

➔Here class PurchaseOrderType is representing the structure of the XSD document.
➔These **classes are called as Binding Classes as these are bonded to the XSD document structure** and the objects of these classes are called binding objects.
➔We can convert the XML document data into a Binding Class Object.
➔**The process of converting an XML to a Binding class Object is called Un-Marshalling**
➔**And the process of converting a Binding Object back into XML representation is called Marshalling.**



➔The above architecture is the generalized architecture which can be applied to any XML Binding API's including JAX-B.

➔Both the types of operations Un-Marshalling and Marshalling are supported JAX-B.
➔**But to perform Un-Marshalling and Marshalling Operations we need binding classes.**
➔So, JAX-B supports two types of operations

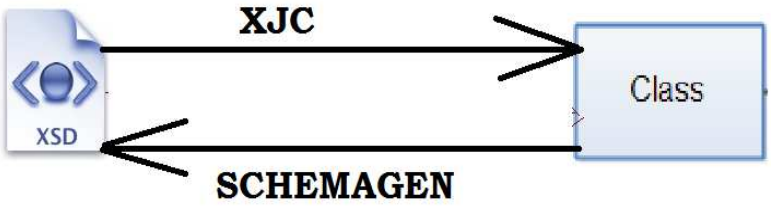
➔One-Time operations.
➔Runtime operations.

One-Time operations

➔ One-Time operations are the operations which would be done only once within the Lifetime of an XSD or Binding class.
➔If we want to do un-marshalling we need binding classes representing the structure of the XSD.
➔ So the programmer has to read the contents of the XSD and based on the number of complex types he needs to create equivalent Java classes.
➔So if complexType is more then it will become time taking task.

➔ So, instead of writing the Binding classes manually, **JAX-B has provided a tool to generated Binding classes from XSD called XJC**. And **to generate XSD from a Binding class there is another tool called Schemagen.**
➔ If I have an XSD document, I need to generate Java classes only once to hold the data, because classes will not hold data they just represents structure.
➔ Instances of those classes hold data of an XML.
➔ Even the reverse would also be same. **As we need to generate the Binding classes or XSD only once** this **process is called one-time operations**.
➔And there are two types of one-time operations.

- ⇨ **XJC – XSD to Java compiler – Generates Binding classes from XSD document**
- ⇨ **Schemagen – Schema generator – Generated XSD document from an Binding class**



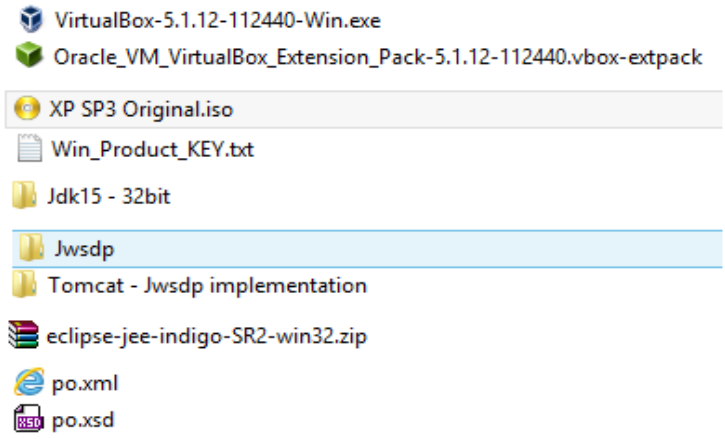
How to use XJC or Schemagen?

➔XJC or Schemagen are the tools that will come when we install JWSDP. **JWSDP** stands for **Java Web Service Developer Pack.**
➔In java 1.5v there is no JWSDP as part of API so to use this we need install JWSDP
But before using JWSDP we have to do some installation process like

➔Install Oracle VM Virtual Box
➔Install XP Operating System

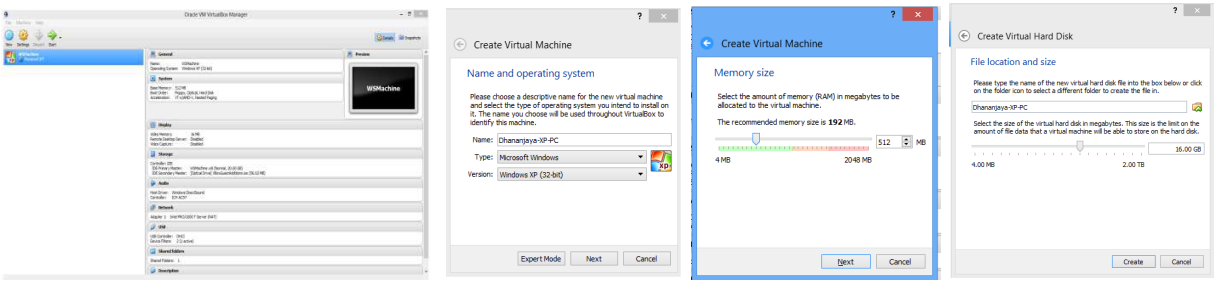
How to install Oracle VM Virtual Box in your system?

Required Software's

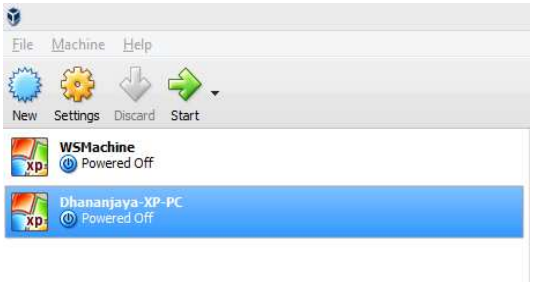


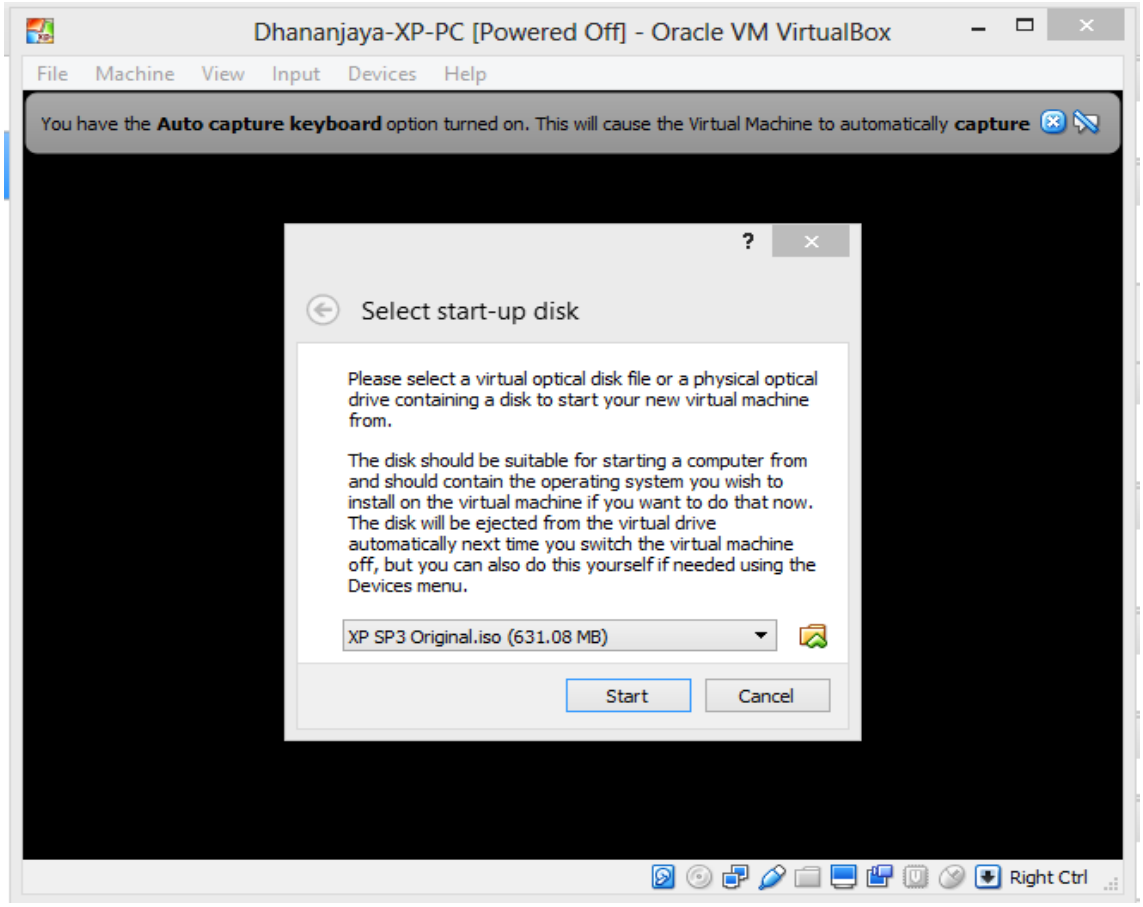
Steps for Minimum requirement

➔Install VirtualBox in your System.
➔New➔select Microsoft Windows=➔Windows XP (32-bit)➔Next➔Memory size=512➔Create Virtual HardDisk now➔VDI(Virtual Disk Image)➔Dynamic Allocated➔File location and Size=16GB

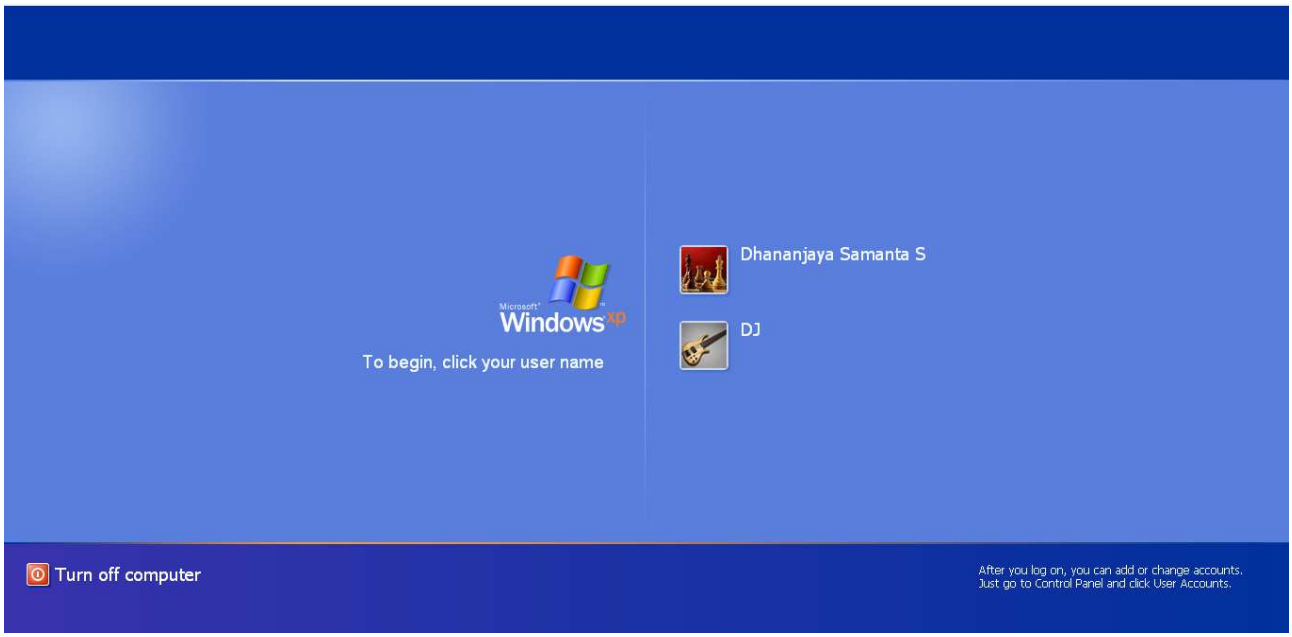


➔Select the XP Machine and Start ➔





➔Select the XP ISO file and start installation of XP OS in your System



➔After OS Installation complete install jdk-**1_5_0_22-windows-i586-p** in your system

➔Then copy the tomcat50-jwsdp and then create a folder 'Apache Software Foundation' inside Program Files and inside that folder paste and extract the tomcat50-jwsdp .jar.

➔Then install **jwsdp-2_0-windows-i586.exe**

➔When we install **JWSDP** by default it would be installed under **c:\Sun\JWSDP-2.0.** Under this directory it will have one more directory called JAXB this is the directory under which all the JAXB related tools and jars would be located.

API

```
C:\Sun\JWSDP-2.0
        |-- JAXB
        | -- lib
        | -- bin
              |-- xjc.bat
              |-- xjc.sh
              |-- schemagen.bat
              |-- schemagen.sh
```

IMPLEMENTATION

```
C:\Program Files\Apache Software Foundation\tomcat50-jwsdp
                                    |-- jaxb
                                        |-- lib
                                            |-- jaxb1-impl.jar
                                            |-- jaxb-api.jar
                                            |-- jaxb-impl.jar
                                            |-- jaxb-xjc.jar
```

➔Set the Environment variables

PATH

- ⇨ C:\Program Files\java\ jdk-1_5_0_22\bin
- ⇨ C:\Sun\JWSDP-2.0\bin

JAVA_HOME

- ⇨ C:\Program Files\java\ jdk-1_5_0_22

---End installation process---

➔Open Indigo Eclipse ➔create a Java Project [JAX-B Project]➔ create a resources folder inside[JAX-B Project] ➔put the po.xsd file inside resources folder.

➔Open command prompt

➔ **cd ...** go to project location

➔**xjc –d src –verbose resources\po.xsd**

➔Refresh the project after that you will see in side src folder of your project class files will be created.

➔But those classes are showing error so you have to add implementation jars in build path.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:to="http://www.ebay.com/sales/types"
        targetNamespace="http://www.ebay.com/sales/types" elementFormDefault="qualified" attributeFormDefault="qualified">
    <xs:element name="purchase-order" type="to:purchase-order-type"/>
    <xs:complexType name="purchase-order-type">
        <xs:sequence>
            <xs:element name="order-item" type="to:order-item-type"/>
            <xs:element name="shipping-address" type="to:shipping-address-type"/>
        </xs:sequence>
    </xs:complexType>                                                                    4
    <xs:complexType name="shipping-address-type">
        <xs:sequence>
            <xs:element name="address-line1" type="xs:string"/>
            <xs:element name="address-line2" type="xs:string"/>
            <xs:element name="city" type="xs:string"/>
            <xs:element name="state" type="xs:string"/>
            <xs:element name="zip" type="xs:string"/>
            <xs:element name="country" type="xs:string"/>
        </xs:sequence>
    </xs:complexType>                                                                    3
    <xs:complexType name="order-item-type">
        <xs:sequence>
            <xs:element name="item" type="to:item-type" minOccurs="1" maxOccurs="unbounded"/>
        </xs:sequence>
    </xs:complexType>                                                                    2
    <xs:complexType name="item-type">
        <xs:sequence>
            <xs:element name="item-code" type="xs:string"/>
            <xs:element name="quantity" type="xs:int"/>
        </xs:sequence>
    </xs:complexType>                                                                    1
</xs:schema>
```

ItemType.java	11-01-2017 09:52	JAVA File	3 KB
ObjectFactory.java	11-01-2017 09:50	JAVA File	3 KB
OrderItemType.java	11-01-2017 09:52	JAVA File	3 KB
package-info.java	11-01-2017 09:50	JAVA File	1 KB
PurchaseOrderType.java	11-01-2017 09:52	JAVA File	4 KB
ShippingAddressType.java	16-01-2017 01:20	JAVA File	6 KB

What does XJC generates

➔When we run XJC.bat by giving XSD as an input along with generating binding classes it generates few other things as explained below.

➔Generates Package based on the Namespace – In an XSD document, it would contain an XSD namespace declaration. Namespaces in XSD similar to packages in Java.

➔So when XJC is generating Binding class for every ComplexType declaration. It would have to even consider mapping the XSD Namespace to package name and should generate the classes under that mapped package (**because the ComplexType declared under targetNamepace are bonded to Namespace, so those classes have to be bonded to packages).**

TargetNamespace in XSD – http://ebay.in/sales/types

create Package Name in Java class

➔Ignore http:// or www

➔Find the first "/" forward slash and extract the substring (ebay.in)

➔Now delimit by "." (dot) and extract tokens in reverse order in.ebay

➔In the remaining string "sales/types" delimit by "/" forward slash and append them to the first constructed package name

➔The resulted package name would become **in.ebay.sales.types.**

➔All the binding classes would be generated into the above package

Package-info.java

➔Package-info.java store this mapping information .
➔This is also generated under the above mapped package only.

```
@javax.xml.bind.annotation.XmlSchema(namespace = "http://www.ebay.com/sales/types")
package com.ebay.sales.types;
```

ObjectFactory.java

➔ These Class Acts as factory class,
➔It create objects of all the Binding classes .
➔It contains methods like **createXXX**() method which return respective binding class objects

```java
package com.ebay.sales.types;

@XmlRegistry
public class ObjectFactory {

  private final static QName _PurchaseOrder_QNAME = new QName("http://www.ebay.com/sales/types", "purchase-order");

   public ObjectFactory() {
   }
   public ItemType createItemType() {
      return new ItemType();
   }

   public ShippingAddressType createShippingAddressType() {
      return new ShippingAddressType();
   }

   public PurchaseOrderType createPurchaseOrderType() {
      return new PurchaseOrderType();
   }

   public OrderItemType createOrderItemType() {
      return new OrderItemType();
   }

   @XmlElementDecl(namespace = "http://www.ebay.com/sales/types", name = "purchase-order")
   public JAXBElement<PurchaseOrderType> createPurchaseOrder(PurchaseOrderType value) {
      return new JAXBElement<PurchaseOrderType>(_PurchaseOrder_QNAME, PurchaseOrderType.class, null, value);
   }
}
```

Which class are called as Binding classes?

The classes which are annotated with JAX-B annotations are called binding classes.

Annotations

@XMLType

➔class level annotation.

➔ It indicated this class is representing an XML Complex Type.

@XMLAccessorType(AccessType.FIELD)

➔class level annotation In conjunction with @XMLType.

➔It denotes how to **map XML elements to Binding** class attributes.

➔With the AccessType.FIELD we are saying the elements names must be matched with binding class attributes names to port the data.

@XMLElement

This annotation is marked at the attributes of the Binding classes to indicate these attributes are representing elements of XML.

What does SCHEMAGEN generates

➔Open command prompt

➔ **cd …** go to project location\bin

➔**schemagen –d . com.ebay.sales.types.PurchaseOrderType**

➔Then go to the project directory then go to bin directory then you will find schemagen generated 2 XSD files linked with each other.

Schema2.XSD

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<xs:schema xmlns:ns1="http://www.ebay.com/sales/types"
           xmlns:xs="http://www.w3.org/2001/XMLSchema" version="1.0">
      <xs:import namespace="http://www.ebay.com/sales/types" schemaLocation="schema1.xsd"/>
      <xs:element name="purchaseOrderType" type="purchase-order-type"/>
      <xs:complexType name="purchase-order-type">
            <xs:sequence>
                  <xs:element ref="ns1:order-item" minOccurs="0"/>
                  <xs:element ref="ns1:shipping-address" minOccurs="0"/>
            </xs:sequence>
      </xs:complexType>
      <xs:complexType name="order-item-type">
            <xs:sequence>
                  <xs:element ref="ns1:item" minOccurs="0" maxOccurs="unbounded"/>
            </xs:sequence>
      </xs:complexType>
      <xs:complexType name="item-type">
            <xs:sequence>
                  <xs:element ref="ns1:item-code" minOccurs="0"/>
                  <xs:element ref="ns1:quantity" minOccurs="0"/>
            </xs:sequence>
      </xs:complexType>
      <xs:complexType name="shipping-address-type">
            <xs:sequence>
                  <xs:element ref="ns1:address-line1" minOccurs="0"/>
                  <xs:element ref="ns1:address-line2" minOccurs="0"/>
                  <xs:element ref="ns1:city" minOccurs="0"/>
                  <xs:element ref="ns1:state" minOccurs="0"/>
                  <xs:element ref="ns1:zip" minOccurs="0"/>
                  <xs:element ref="ns1:country" minOccurs="0"/>
            </xs:sequence>
      </xs:complexType>
</xs:schema>
```

schema1.XSD

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           xmlns:ns1="http://www.ebay.com/sales/types"
           targetNamespace="http://www.ebay.com/sales/types" version="1.0">
      <xs:element name="address-line1" type="xs:string"/>
      <xs:element name="address-line2" type="xs:string"/>
      <xs:element name="city" type="xs:string"/>
      <xs:element name="country" type="xs:string"/>
      <xs:element name="item" type="item-type"/>
      <xs:element name="item-code" type="xs:string"/>
      <xs:element name="order-item" type="order-item-type"/>
      <xs:element name="quantity" type="xs:int"/>
      <xs:element name="shipping-address" type="shipping-address-type"/>
      <xs:element name="state" type="xs:string"/>
      <xs:element name="zip" type="xs:string"/>
</xs:schema>
```

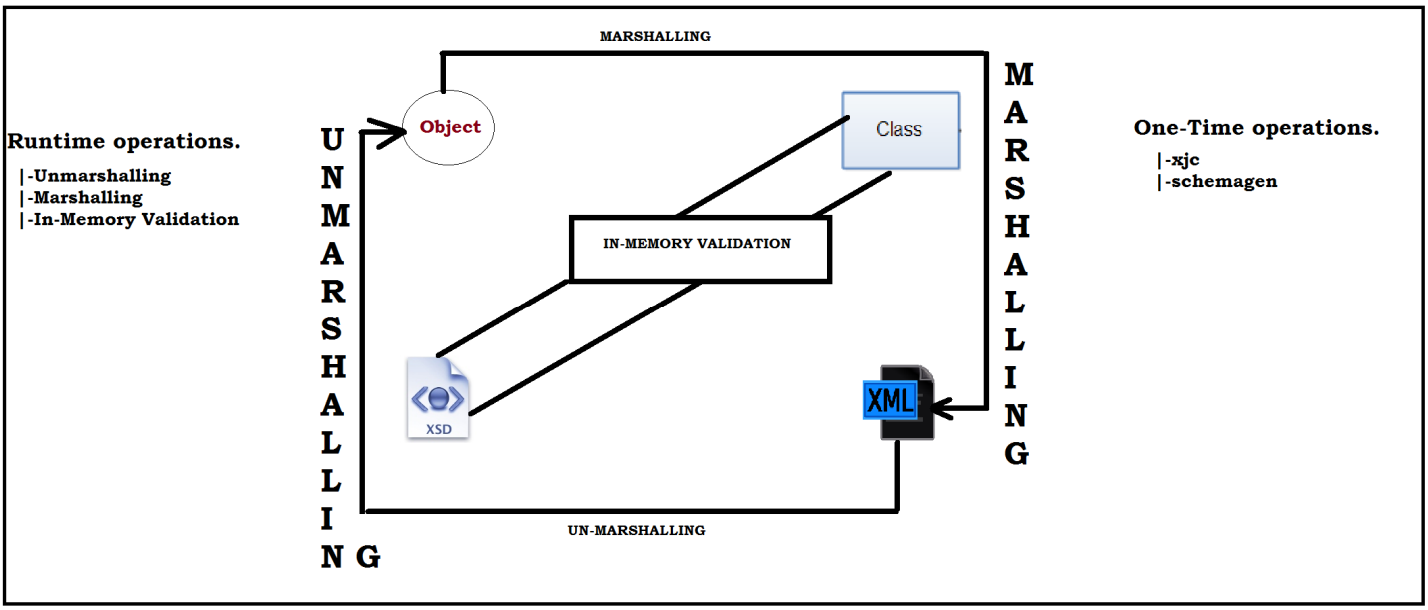
Runtime Operation

➔JAX-B supports three runtime operations.

➔ Un-Marshalling – The process of converting XML to Java Object.

➔ Marshalling – The process of converting Java Object with data to its Equivalent XML.

➔ In-Memory Validation – Before converting an XML document to Java Object first we need to validate whether the XML conforms to XSD or not would be done using In-Memory Validation.



➔**If we want to perform runtime operations, first you need to do one-time operation.**

➔**Unless you have binding classes or XSD equivalent for any existing binding class, you can't work on runtime operations.**

UN-MARSHALLING with In-Memory Validate

➔Now after one-time operation we have binding classes but having the **binding classes will not automatically converts the XML** document into Object of these classes, **rather we need a mediator who would be able to read the contents of the XML**, and identify the respective binding class for that and creates objects to populate data. **This will be done by a class called "Unmarshaller".**

➔As JAX-B is an API, so Unmarshaller is an interface defined by JAX-B API
➔And its JAX-B API implementations (JAX-B RI or JaxMe) contains the implementation class for this interface.
➔The object of Unmarshaller has a method unmarshal().
➔ **unmarshal() method takes the input as XML document and first check for Well-formness.**

➔Then it reads the elements of the XML and tries to identify the binding classes for those respective elements based on **@XMLType and @XMLElement** annotations and converts them into the Objects.

➔We have two problems here

➔How to created Un-marshaller as we don't know implementation class

➔Where does un-marshaller should search for identifying a binding class (is it in entire JVM memory or classpath?)

➔As we don't know how to identify the implementation of Unmarshaller interface, **we have a factory class called JAXBContext** who will finds the implementation class for Unmarshaller interface and would instantiate.

➔**It has a method createUnmarshaller() which returns the object of unmarshaller.**

```
//UnMarshalling
        Unmarshaller unmarshaller=context.createUnmarshaller();
        JAXBElement<PurchaseOrderType> po=
                (JAXBElement<PurchaseOrderType>)unmarshaller.unmarshal(new     File("C:/workspace/JAX-B/resources/po.xml"));
        System.out.println(po.getValue());
```

➔ While creating the JAXBContext, we need to supply any of the three inputs.

➔Package name containing the binding classes

(Or)

➔ObjectFactory.class reference as it knows the binding classes.

(Or)

➔Individual Binding classes as inputs to the JAXBContext.

➔ Now when we call the Unmarshaller.unmarshal(new File("po.xml")) method, the Unmarshaller will searches with the classes loaded in the JAXBContext and tries to identifies the classes for an XML element based on their annotations.

Un-Marshalling Example

```
package com.jaxb.unmarshaller;

import java.io.File;
import javax.xml.bind.*;
import javax.xml.validation.SchemaFactory;
import org.xml.sax.SAXException;
import com.ebay.sales.types.PurchaseOrderType;
/**
 *
 * @author Dhananjaya Samanta S
 * program for validating the XML and UNmarshalling
 */
public class UnMarshallerDemo {

    public static void main(String[] args) throws JAXBException, SAXException {
        //Loading MetaData and factory class for UnMarshalling
        JAXBContext context=JAXBContext.newInstance("com.ebay.sales.types");

        //validating
        SchemaFactory sfactory=SchemaFactory.newInstance("http://www.w3.org/2001/XMLSchema");
        sfactory.newSchema(new File("C:/Documents and Settings/Dhananjaya Samanta S/workspace/JAX-B/resources/po.xsd"));

        //UnMarshalling
        Unmarshaller unmarshaller=context.createUnmarshaller();
        JAXBElement<PurchaseOrderType> po=(JAXBElement<PurchaseOrderType>)
            unmarshaller.unmarshal(new File("C:/Documents and Settings/Dhananjaya Samanta S/workspace/JAX-B/resources/po.xml"));
        System.out.println(po.getValue());
    }
}
```

Marshalling

➔ Marshalling is the process of converting the Object data to XML.
➔For performing Marshalling also we need to first of all have the Binding classes.
➔Once we have the Binding classes, we need to create the Object of these classes and should populate data in them.
➔ These objects can be marshalled to XML using the Marshaller

```
//Marshalling    (Object to XML)
    JAXBContext jContext=JAXBContext.newInstance("com.ebay.sales.types");
    Marshaller marshaller=jContext.createMarshaller();
    marshaller.marshal(purchaseOrder,System.out);
```

➔As we don't know how to identify the implementation of Marshaller interface, **we have a factory class called JAXBContext** who will finds the implementation class for Marshaller interface and would instantiate.
➔It has a method createMarshaller() which returns the object of Marshaller.

➔ While creating the JAXBContext, we need to supply any of the three inputs.
➔Package name containing the binding classes
(Or)
➔ObjectFactory.class reference as it knows the binding classes.
(Or)
➔Individual Binding classes as inputs to the JAXBContext.

➔But Before calling **marshal(purchaseOrder,System.out)** we have to add an Annotation in class level called **@XmlRootElement in PurchaseOrderType other wise you will get RE:Unable to marshal type...Missing an annotation @XmlRootElement**

```
package com.ebay.sales.types;

@XmlAccessorType(AccessType.FIELD)
@XmlType(name = "purchase-order-type", propOrder = {"orderItem","shippingAddress"})

@XmlRootElement
public class PurchaseOrderType {

   @XmlElement(name = "order-item", namespace = "http://www.ebay.com/sales/types")
   protected OrderItemType orderItem;
   @XmlElement(name = "shipping-address", namespace = "http://www.ebay.com/sales/types")
   protected ShippingAddressType shippingAddress;

   //getters & setters
}
```

Marshalling Example

```
package com.jaxb.marshaller;
import java.io.File;
import javax.xml.bind.*;
import javax.xml.validation.*;
import org.xml.sax.SAXException;
import com.ebay.sales.*;

public class MarshallerTest {
	public static void main(String[] args) throws JAXBException, SAXException {

		ShippingAddressType shippingAddress=new ShippingAddressType();
		shippingAddress.setAddressLine1("505");
		shippingAddress.setAddressLine2("Mytrivanam");
		shippingAddress.setCity("Hyderabad");
		shippingAddress.setState("Telengana");
		shippingAddress.setZip("700038");
		shippingAddress.setCountry("India");

		ItemType items=new ItemType();
		items.setItemCode("c111");
		items.setQuantity(50);

		ItemType items1=new ItemType();
		items1.setItemCode("c22");
		items1.setQuantity(45);

		OrderItemType orderItem=new OrderItemType();
		orderItem.getItem().add(items);
		orderItem.getItem().add(items1);

		PurchaseOrderType purchaseOrder=new PurchaseOrderType();
		purchaseOrder.setOrderItem(orderItem);
		purchaseOrder.setShippingAddress(shippingAddress);

		JAXBContext jContext=JAXBContext.newInstance("com.ebay.sales.types");
		Marshaller marshaller=jContext.createMarshaller();
		marshaller.marshal(purchaseOrder,System.out);
	}
}
```

IN-Memory Validation

➔ Using In-Memory validation we can validate XML is valid against the XSD While performing the Un-Marshalling or Marshalling.
➔ An XML can be validated against an XSD document, so we need to read the XSD document and should represent it in a Java Object called Schema.
➔ But Schema is an interface and to create the Object of interface, we need factory called **SchemaFactory**.
➔But schema factory can created various types of XSD documents, so we need to provide the (standard) grammar of XSD document which you want to load or created.

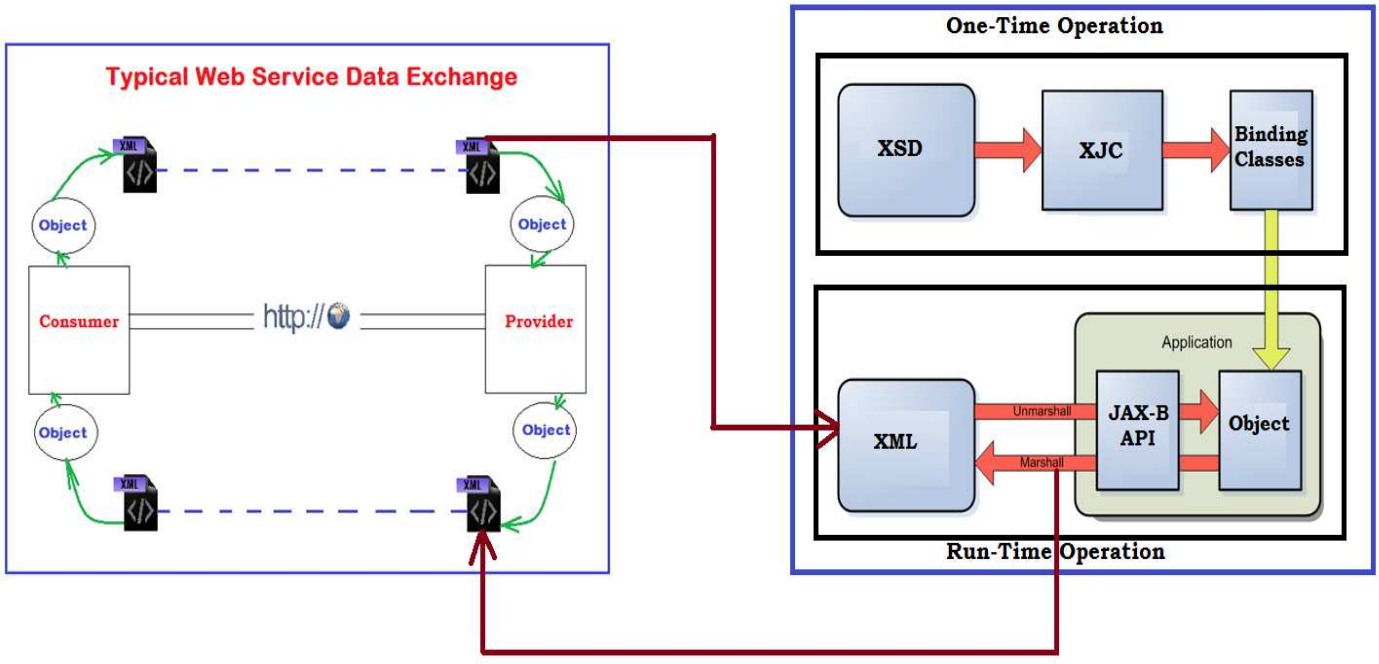
```
SchemaFactory sFactory =
SchemaFactory.newInstance(XMLConstants.W3C_XML_SCHEMA_URI);
```

➔Now we need to load the XSD document into the Schema class object.

```
Schema poSchema = sFactory.newSchema(new File("po.xsd"));
```

➔Once the schema object is created holding the structure of XSD.
➔We need to give it to Unmarshaller before calling the unmarshal().
➔So, that Unmarshaller will validate the input XML with this supplied Schema Object and then performs Un-marshalling.
➔Otherwise Un-marshaller will throw an exception, indicating the type of validation failure.

JAX-B Internally Made for converting XML to Object and Object to XML



JAX-RPC