JSPIDER
BASAVANGUDI
BANGALORE

WEB SERVICES – DAY 2
URI (URL & URN), XML, JSON, JAXB, JACKSON

| Praveen D

Table of Contents

URI, URL and URN

- Uniform Resource Identifier (URI) is a string of characters used to

- identify a resource using name or
- Locate a resource in the network

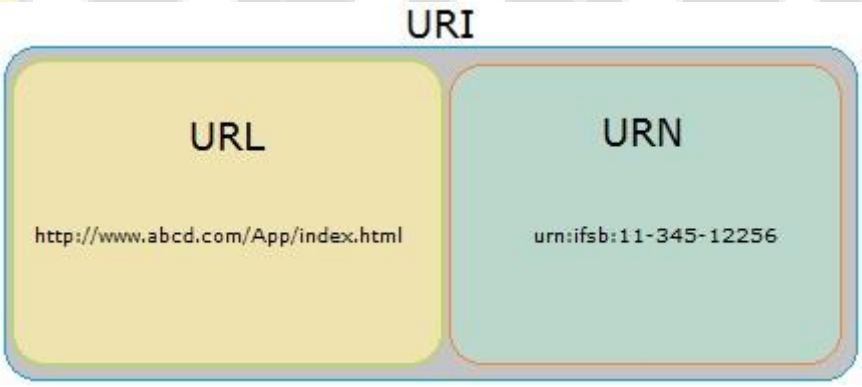
- A URI identifies a resource either by location, or a name, or both. A URI has two specializations
  1. URL (Uniform Resource Locator) and
  2. URN (Uniform Resource Name)

- URN ONLY identifies the resource and does not let us know availability of the resource. A URN has to be of this form "urn:"

- URL that specifies where an identified resource is available and the mechanism for retrieving it. URL does not have to be HTTP URL (http://), a URL can also be (ftp://) or (smb://) or (jdbc:)

- For example,
  - A URN is similar to a person's name, while
  - A URL is like a street address.
  - The URN defines something's identity, while the URL provides a location.
  - Essentially, "what" vs. "where"

- To put it differently,
  - A URL is a URI
  - A URN is a URI
  - but URNs and URLs are different, A URI is not necessarily a URL



- Few Examples:-

URL: ftp://ftp.is.co.za/rfc/rfc1808.txt
URL: http://www.ietf.org/rfc/rfc2396.txt
URL: ldap://[2001:db8::7]/c=GB?objectClass?one
URL: mailto:John.Doe@example.com
URL: news:comp.infosystems.www.servers.unix
URL: telnet://192.0.2.16:80/
URN (not URL): urn:oasis:names:specification:docbook:dtd:xml:4.1.2
URN (not URL): urn:isbn:0-486-27557-4

Web URL (Web Uniform Resource Locator's)

- Web URL, uniquely identifies a particular web resource inside a web application

- In other words, every web resource should have its unique address in the form of Web URL

- Max. number of characters allowed in Web URL is around 2000 characters (exact number depends on Browser. For ex, IE supports 2048 characters)

- Web URL Structure:
  Protocol://Domain:Port/Path?QueryString#FragmentID

- Protocol in case of Web URL is always http or https

- Domain Name uniquely identifies a computer in a network in which web application is present. It can be Computer Name/DNS Name (preferred) or IP address

- Port number in Web URL uniquely identifies web server application

- Default port number for HTTP is 80 & HTTPS is 443

- In Tomcat Webserver, default port number for HTTP is changed from 80 to 8080 and default port number for HTTPS is changed from 443 to 8443

- Path is the full path of the web resource at web application side.
  - It consists of, Web Application Name / Configured URL of a Resource
  - "Web Application Name" uniquely identifies One web application inside webserver
  - "Configured URL" uniquely identifies web resource inside that web application

- Query String is a name & value string pair which passes information in the form of name=value pair to web resources. In URL, It's an optional information and if present, it starts with question mark followed by one or more name=value pair which are separated by an ampersand(&)

- A Fragment ID or Fragment Identifier, as the name implies, it refers to a particular fragment / a section within a web page
- Matrix Parameters are a set of "name=value". They can be present anywhere in URL (generally used with **path**) & URL can consist of N number of Matrix parameters but they should be separate by a semi colon ";"

- The important difference between Query Parameters & Matrix Parameters is that,
  - Matrix Parameters apply to a particular path element while
  - Query Parameters apply to the request as a whole
  - This comes into play when making a complex REST-style query to multiple levels of resources and sub-resources

- **Note:-**
  - Apart from Domain Name rest of the components of Web URL are Optional
  - Few Examples:-

http://www.google.com/search?q=Praveen
https://www.google.co.in/search?q=ABC&sitesearch=www.youtube.com
http://www.example.com/res/categories;name=foo/objects;name=green/?page=1

eXtensible Mark-up Language (XML)

- XML is "Programing Language & Platform Independent Language" which helps to store and transport data
- Different Applications which are developed using different technologies can Transfer the Data among themselves with the help of XML
- As the name implies it's an extension of HTML & hence XML looks similar to HTML but it's not a HTML
- XML has User-defind Tags. XML tags are also called as "elements"
- XML Elements are "Case Sensitive"
- XML is "Strictly Typed" Language hence,
  - For every element data, "data-type" should be defined,
  - every opening element should have corresponding closing element and
  - also XML elements must be properly nested/closed

  Ex:

  ```
  <employee>
        <name>Praveen</name>
  </employee>
  ```

  Note:-

  In the above example first you should closed </name> & then </employee> but in HTML it's not mandatory. For example, <B><U><I>My Text</U></I></B> works perfectly fine
- Below line is called as "XML prolog", which is optional. If it exists, it must be the First Line of XML

  <?xml version="1.0" encoding="UTF-8" ?>
- The syntax of XML comment is similar to that of HTML

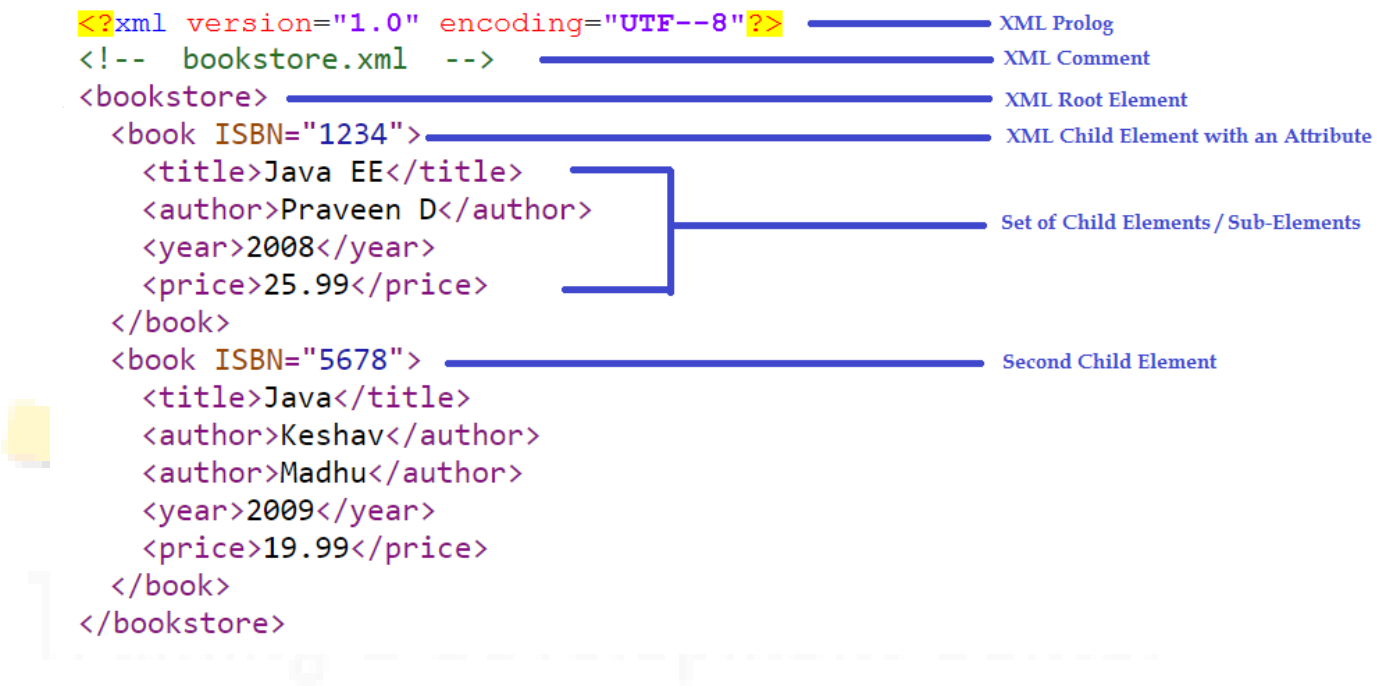
  <!-- This is a comment -->
- File extension of XML is ".xml"
- MIME type (Content Type) of XML is "application/xml"

1. XML Structure

- Like HTML, XML follows a Tree Structure

- An XML tree starts at a "root element" and branches from "root element" will have "child elements"
- XML Consists of "Only One" root element which is parent of all other elements
- "child elements" can have "sub elements / child elements"
- Structure

```
<root>
 <child>
  <subchild>.....</subchild>
 </child>
</root>
```



2. Entity References

- Some characters have a special meaning in XML. If you place a character like "<" inside an XML element, it will generate an error because it represents the start of a new element
  Ex: <message>salary<1000</message>
- To avoid this error, we can replace the "<" character with an "entity reference" as shown below
  <message>salary **<** 1000</message>
- There are 5 pre-defined entity references in XML:

<	<	less than
>	>	greater than
&	&	ampersand

&apos;	'	apostrophe
&quot;	"	quotation mark

3. PCDATA: Parsed Character Data

- Text between start-element and end-element is called as PCDATA which will be examined by the parser

Example:-
<employee>Praveen</employee>
The string "Praveen" is considered as PCDATA

4. CDATA: Character Data

- W.K.T special characters (such as "<", "&") must be referenced through pre-defined entities
- If XML data contain many special characters, it is cumbersome to replace all of them. Instead we can use "CDATA (character data) section"
- A CDATA section starts with the following sequence:

  <![CDATA[

  and ends with the next occurrence of the sequence:

  ]]>

  All characters enclosed between these two sequences are interpreted as characters
- The XML parsers ignores all the mark-up within the CDATA section.

  Example: -

  <employee>Praveen</employee>
  the start and end "employee" elements are interpreted as mark-up. However, if written like this:

  <![CDATA[<employee>Praveen</employee>]]>

  then the parsers interprets the same as if it had been written like this:

  <employee>Praveen</employee>

5. XML Elements

- XML element is everything from (including) the element's start tag to (including) the element's end tag

- An element can contain:
  1. data
  2. Attributes
  3. other elements OR
  4. All of the above

- In the above example
  - <title>, <author>, <year>, and <price> have text content
  - <bookstore> and <book> have element contents
  - <book> has an attribute (ISBN="------")

- An element with no content is said to be "empty". In XML, we can indicate an empty element like this
  <element></element>
  OR
  <element />

- Empty elements can have attributes <book ISBN="5678" />

- If data present between elements consist of white spaces then they are considered in XML. However HTML truncates multiple white-spaces to one single white-space

6. XML Elements Naming Rules

- they are case-sensitive
- they cannot contain spaces
- they must start with a letter or underscore
- they are cannot start with the letters like xml or XML or Xml etc.,
- they can contain letters, digits, hyphens, underscores, and periods
- Any name can be used, no words are reserved (except xml)

Best Naming Practices

- Avoid "." and ":"

- Create descriptive names, like
  <person>, <firstname>, <lastname>

- Create short and simple names, like
  <book_title> not like this: <the_title_of_the_book>

- Non-English letters are perfectly legal in XML but avoid them

7. XML Attributes

- Like HTML, XML elements can also have attributes

- Attributes are designed to contain data related to a specific element

- XML Attributes Must be Quoted either single or double quotes can be used
  Ex:
  <person gender="female">
  OR
  <person gender='female'>

- If the attribute value itself contains double quotes then we can use single quotes
  Ex:
  <person name='Praveen "Bangalore" D'>
  OR
  <person name='Praveen "Bangalore" D'>

8. XML Elements v/s Attributes

Example 1:-

```
<person gender="male">
        <name>Praveen</name>
</person>
```

Example 2:-

```
<person>
        <gender>male</gender>
        <name>Praveen</name>
</person>
```

Note:

- In Example 1 gender is an attribute &
- In Example 2 gender is an element
- Both examples provide the same information
- There are no rules about when to use attributes or when to use elements in XML

When to avoid XML Attributes?

- Attributes cannot contain multiple values but Elements can
- Attributes cannot contain tree structures but Elements can
- Attributes are not easily expandable for future changes but Elements can

9. XML Schema's

- W.K.T XML helps us to store & transfer the data

- When sending data from one application to an another, it is essential that both applications have the same "expectations / agreement" about the content/data
- for example, A date like "03-11-2004"
  - in some countries, be interpreted as 3rd November and
  - in other countries as 11th March

- With XML Schemas, the sender application can describe the data in a way that the receiver application will understand

- Schema is nothing but a "Structure". It is a formal description of structure of an XML.
  - i.e., which elements are allowed,
  - which elements must be present,
  - which elements are optional,
  - the sequence and relationship of the elements, etc.,

- For example,
  - abc@gmail.com is a Valid Email ID. However
  - abc#gmail is Invalid because there is "NO @ and ."
  - hence email schema looks something like some-name@domain-name.com
- Schema "does not validate the data" instead "it validates the structure"

- There are two ways to define a Schema for XML
  1. Document Type Definition (DTD)
  2. XML Schema Definition (XSD)

1) XML Document Type Definition (DTD)

- A DTD defines the structure and the legal elements and attributes of an XML document

- An application can use a DTD to verify that XML data is valid

- There are 2 ways to declare the DTD
  1. An Internal DTD Declaration
  2. An External DTD Declaration

- An Internal DTD Declaration has the following syntax:

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

XML document with an internal DTD

```
<?xml version="1.0"?>
<!DOCTYPE note [
<!ELEMENT note (to,from,heading,body)>
<!ELEMENT to (#PCDATA)>
<!ELEMENT from (#PCDATA)>
<!ELEMENT heading (#PCDATA)>
<!ELEMENT body (#PCDATA)>
]>
<note>
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Don't forget me this weekend</body>
</note>
```

- A DTD can also be stored in an external file. An XML can reference an external DTD via the following syntax:

  <!DOCTYPE root-element SYSTEM "DTD-filename">

XML document with a reference to an external DTD

```
<?xml version="1.0"?>
<!DOCTYPE note SYSTEM "note.dtd">
<note>
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
</note>
```

"note.dtd"

```
<!ELEMENT note (to,from,heading,body)>
<!ELEMENT to (#PCDATA)>
<!ELEMENT from (#PCDATA)>
<!ELEMENT heading (#PCDATA)>
<!ELEMENT body (#PCDATA)>
```

2) XML Schema Definition (XSD)

- XSD also describes the structure, legal elements and attributes for an XML
- It defines,
  - the elements and attributes that can appear in XML
  - the number of and also the order of child elements
  - data types for elements and attributes
  - default and fixed values for elements & attributes
- One of the greatest strength of XML Schemas is the support for data types
- For Example, the following is an example of a date declaration in XSD:
  <xs:element name="start-date" type="xs:date"/>

  it defines the structure/format of the Date as "YYYY-MM-DD"

  An element in XML might look like <start-date>2002-09-24</start-date>
- Another great strength about XML Schemas is that they are written in XML
- Hence XSD's are extensible so, we can
  - Reuse Schema in other Schemas
  - Create your own data types derived from the standard types
  - Reference multiple schemas in the same document

NOTE:

- Functionality wise both XSD & DTD similar in nature but XSD's are more sophisticated compared to DTD

- In other words, DTD provides less control on XML structure whereas XSD provides more control

- Hence XSD's preferred over DTD's

- Without an XSD/DTD, an XML need only follow the rules for being well-formed

- With an XSD/DTD, an XML must adhere to additional constraints placed upon the names and values of its elements and attributes in order to be considered valid

10. Differences between DTD & XSD

DTD	XSD
DTD's are written in Mark-up Language	XSD's are written in XML
DTD is not extensible i.e. We cannot inherit one DTD into an another	XSD is extensible. We can inherit one XSD into an another
DTD doesn't support data types (limited to string)	XSD supports data types for elements and attributes
DTD doesn't define order for child elements	XSD defines order for child elements
DTD's occurrence indicator is limited to 0, 1 and many; cannot support a specific number such as 8	XSD can support a specific number
DTD doesn't support namespace	XSD supports namespace
We cannot inherit one DTD into an another	We can inherit one XSD into an another
DTD provides less control on XML structure	XSD provides more control on XML structure

11. Parsing XML Documents (XML Parsers)

- To process the data contained in XML documents, we need to write a application program (in any programming language such as Java/C/C++, etc)

- The program makes use of an XML parser to tokenize and retrieve the data from the XML documents

- An XML parser is the software that sits between the application and the XML documents to shield the application developer from the details of the XML syntax.

- The parser reads a raw XML document, ensures that is well-formed, and may validate the document against a DTD or XSD

- There are two standard APIs for parsing XML documents:
    1. SAX (Simple API for XML)

2. DOM (Document Object Model)

- The JAXB provides a common interface for creating, parsing and manipulating XML documents using the standard SAX, DOM and XSLTs

Java Architecture for XML Binding (JAXB)

- JAXB is part of the JDK (from 1.6 onwards) & "it is the most often used API to process XML "
- JAXB is a Java API helps us to convert Java Object to XML & vice-versa
- The Process of converting Java Object to XML is called as "Marshalling" OR "Serialization"
- The Process of converting XML to Java Object is called as "Unmarshalling" OR "Deserialization"
- "javax.xml.bind.*" is the package representation of JAXB
- JAXB also helps us to generate XSD using Java Objects (with the help of "schemagen" commond)
- It helps us to Generate Java Objects using XSD (with the help of "xjc" commond)
- The JAX-RS API uses JAXB to convert the XML data from the request into a Java Object & vice-versa

1. Commonly Used/Basic Annotations in JAXB

- W.K.T Java Annotations provide metadata on
  - packages
  - classes
  - fields and
  - methods

- Also each annotation type has its own set of "annotation elements"
- JAXB can be used in two ways
  - Creating our own Java Beans using JAXB Annotations
  - Generating & Using Java Beans using XSD

1. @XmlRootElement

- A class that describes an XML element that is to be a top-level element, should be annotated with XmlRootElement
- This annotation should be used with a Class Name

- When a top level class is annotated with the @XmlRootElement annotation, then its value is represented as XML element in an XML

- It has 2 Optional Annotation Elements

1. name:-

- It's the name of the XML root element
- Default name is derived from the class name

2. namespace:-

- "namespace" name of the XML element
- Default is the empty namespace

2. @XmlElement

- This annotation helps to define an element in an XML

- This Annotation should be used with Class Variables (preferred) / Getter Methods

- By default JavaBeans Property Names are used as element names

- The "name" element of this annotation defines name for an Element in XML

3. @XmlAttribute

- Specifies the attribute for XML element

- This Annotation should be used with Class Variables (preferred) / Getter Methods

- By default JavaBeans Property Names are used as Attribute names

- The "name" element of this annotation defines name for an Element in XML

4. @XmlType(propOrder = { "field2", "field1",.. })

- This annotation should be used with a Class Name

- This annotation with element "propOrder" allows to define the order in which the fields are written in an XML file

5. @XmlElementWrapper

- XmLElementWrapper generates a wrapper element around XML representation. Its used with collections (like array, Lst, Set, etc.,)

- This is primarily intended to be used to produce a wrapper XML element around collections

- By default JavaBeans Property Names are used as Element Wrapper name

- The "name" element of this annotation defines name for an Element in XML

6. @XmlAccessorType

- This annotation defines the way class level annotation needs to be treated
  - if @XmlAccessorType(XmlAccessType.FIELD) then
    Class Variables should have JAXB Annotations
  - if @XmlAccessorType(XmlAccessType.PROPERTY) then
    Class Getter Methods should have JAXB Annotations
  - if @XmlAccessorType(XmlAccessType.NONE) then
    either Class Getter Methods/Class Variables (one of them) should have JAXB Annotations

- This annotation should be used with a Class Name along with @XmlRootElement

- If we have JAXB annotations for Class Getter Methods then this annotation can be avoided

2. JAXB - Generating XSD using JAXB Annotated Class

Command:
schemagen -cp <Class_Files_Location>
<Java_File_Location_Which_acts_as_Root_Element>

Note: <Class_Files_Location> should point the beginning of Package not inside the package

Ex:-
C:\Users\Praveen> schemagen -cp
E:\GitHub\java_workspace\myjavaapp\target\classes
E:\GitHub\java_workspace\myjavaapp\src\main\java\com\dyasha\myjavaapp\jaxb\College.java

Output: - Note: Writing C:\Users\Praveen\schema1.xsd

C:\Users\Praveen> -----> you can be in any path

E:\GitHub\java_workspace\myjavaapp\target\classes -----> should point the beginning of Package not inside the package

E:\GitHub\java_workspace\myjavaapp\src\main\java\com\dyasha\myjavaapp\jaxb\College.java -----> Java File location which acts as Root Element

Writing C:\Users\Praveen\schema1.xsd -----> Schema can be found in the same path where you're present in the command prompt

3. JAXB - Generating Java Beans/POJO's using XSD

Command: xjc <XSD_File_Location>

Note:

- Generated Classes will be present inside the folder by name "generated"
- This folder will be present under the path where we run this command
  Ex:- c:\jaxb>xjc d:\jaxb\schema1.xsd
  then "generated" folder along the corresponding classes will be in "c:\jaxb" folder

- We can also provide the desired package name to place the generated classes
  Ex:-
  C:\jaxb>xjc C:\Users\Praveen\schema1.xsd -p com.dyasha.myjavaapp.jaxb.xsd

  Output:-
  parsing a schema...
  compiling a schema...
  com\dyasha\myjavaapp\jaxb\xsd\College.java
  com\dyasha\myjavaapp\jaxb\xsd\ObjectFactory.java
  com\dyasha\myjavaapp\jaxb\xsd\StudentInfo.java

4. JAXB Marshalling [Converting Object into XML]

Steps:

1. Create a Java Beans with JAXB annotations or Using XSD generate the classes
2. Create the JAXBContext object
3. Create the Marshaller object
4. Invoke setProperty method on Marshaller object by certain values (Optional)
5. Invoke the marshal method on Marshaller object

5. JAXB Unmarshalling [Converting XML into Object]

Steps:

1. Create a Java Beans with JAXB annotations or Using XSD generate the classes
2. Create the JAXBContext object
3. Create the Unmarshaller object
4. Invoke the unmarshal method
5. Invoke getter methods of Java Bean to access the data

6. javax.xml.bind.JAXBContext

- The JAXBContext class provides the client's entry point to the JAXB API

- It is an Abstraction layer in JAXB which helps us to perform basic operation of JAXB such as marshal, unmarshal and validate

- It's an abstract class doesn't extend any other JAXB related Class i.e.
  `public abstract class JAXBContext extends Object`

- A client application normally obtains new instances of this class using newInstance() method

  Syntax:

```
public static JAXBContext
newInstance(Class... classesToBeBound)
```

```
throws JAXBException
```

- There should be ONLY ONE object of "JAXBContext" should exists per application. In other words it should be accessed in "Singleton" way. NOTE: "JAXBContext" is not Singleton in nature

7. javax.xml.bind.Marshaller

- Marshaller object is responsible for governing the process of serializing Java content trees back into XML data
- It provides the basic marshalling methods, to convert Java Objects to XML
- It's an Interface & an instance of Marshaller is obtained by invoking "createMarshaller()" method on JAXBContext Object

Syntax

```
Marshaller JAXBContext.createMarshaller()
throws JAXBException
```

8. javax.xml.bind.Unmarshaller

- Unmarshaller object is responsible for governing the process of deserializing XML data into newly created Java content trees, optionally validating the XML data
- It provides lot of overloaded unmarshalling methods, to convert XML to Java Objects
- It's an Interface & an instance of Unmarshaller is obtained by invoking "createUnmarshaller()" method on JAXBContext Object

Syntax

```
Unmarshaller JAXBContext.createUnmarshaller()
throws JAXBException
```

9. JAXB Summary

Annotations Used:

1. @XmlRootElement
2. @XmLElement
3. @XmlAttribute
4. @XmLElementWrapper

5. @XmlType
6. @XmlAccessorType

Java Classes Used:

1. JAXBContext (Abstract Class)
2. Marshaller (Interface)
3. Unmarshaller (Interface)
4. JAXBException (Concreate Class)

JavaScript Object Notation [JSON]

1. Examples of Data Formats:

String

```
String str1 = "123praveen200.12";
String str2 = "EmpID=123|EmpNM=praveen|EmpSal=200.12";
```

XML

```
<employee>
    <emp-id>123</emp-id>
    <emp-name>Praveen</emp-name>
    <emp-salary>200.12</emp-salary>
</employee>
```

JSON

```
{"EmpID":123, "EmpNM":"Praveen", "EmpSal":200.12}
```

- Like XML, JSON also is a "Programing Language & Platform Independent Language" which helps to store and transport data
- However compared to XML, it's a lightweight, easy for applications to parse and generate by avoiding complicated parsing and translations
- JSON is a "text format" but uses conventions that are familiar to programmers of the C-family of languages (C, C++, C#, Java, JavaScript, Perl, Python, etc.,). Hence JSON is an "ideal data interchange language"
- JSON, as the name implies, which consists of data similar to "Object Notation of JavaScript". It's an extension of JavaScript scripting language and this format was specified by "Douglas Crockford in 2006"
- Hence if we receive data from a server in JSON format, we can directly use it like any other JavaScript object
- The filename extension of JSON is ".json"
- MIME type (Content Type) of JSON is "application/json"

2. JSON Syntax

JSON syntax is derived from JavaScript object notation syntax:
- Data is in "name:value" pairs
- Data is separated by "commas"
- "Curly braces" hold objects
- "Square brackets" hold arrays

i. JSON Data

JSON data is written as name/value pairs. A name/value pair consists of

- a field name (**Should be** in double quotes)
- followed by a colon
- followed by a value

Ex: "employee-name" : "Praveen D"

ii. JSON Values

- In JSON, values must be one of the following data types
  1. String
  2. Number
  3. Boolean
  4. NULL
  5. an Object (JSON object)
  6. an Array

- In JSON,
  - String values must be written with double quotes
  - Numbers must be an integer/decimal values
  - Boolean values must be true/false
  - JSON NULL values must be null

- **Ex:-**

```
{ "name":"Praveen D",
  "age":33,
  "isEmployed":true,
  "girlFriend":null
}
```

iii. JSON Objects

- Values in JSON can be objects

- JSON Objects are
  - surrounded by curly braces {}
  - JSON object data is written in "key:value" pairs
  - Each "key:value" pair is separated by a comma
  - Keys must be String and Values must be a valid
  - JSON data type (String, Number, Object, Array, Boolean or null)

- **Ex:-**

```
 {
   "employee":{ "name":"Praveen D",
                "age":33,
                "isEmployed":true,
                "girlFriend":null
              }
 }
```

- Values in a JSON object can be another JSON object

```
{
```

```
        "employee": {
                                "name":"Praveen D",
                                "age":33,
                                "isEmployed":true,
                                "girlFriend":null,
                                "cars": {
                                        "car1":"GM",
                                        "car2":"BMW",
                                        "car3":"Audi"
                                        }
                        }
}
```

iv. JSON Arrays

- Values in JSON can be arrays

- JSON Arrays are
  - surrounded by "Square Brackets []"
  - JSON Arrays values is separated by a comma
  - Array values must be a valid JSON data type
  - (String, Number, Object, Array, Boolean or null)

- Example 1:-

```
{
 "employees":[ "Praveen", "Rekha", "Malleshwar" ]
}
```

- Example 2:-

```
{
 "name":"Praveen",
 "age":33,
 "cars":[ "GM", "BMW", "Audi" ]
}
```

- Values in an array can also be another array, or even another JSON object:

```
{
   "name":"Praveen",
   "age":33,
   "cars": [
                { "name":"GM",
                  "models":[ "Aveo", "Beat", "Cruze"  ]
                },
                    { "name":"Audi",
                      "models":[ "A3", "A7" ]
                    }
            ]
 }
```

JSON v/s XML

- Both JSON and XML can be used to store & get the data from a web server

- However,
  - XML has to be parsed with an XML parser
  - JSON can be parsed by a standard JavaScript function

- Hence, XML is much more difficult to parse than JSON
  JSON is a ready-to-use JavaScript object

- XML Data cannot consist of Arrays
  JSON Data can consist of Arrays

- In General,
  - Web Applications interact with each other by exchanging data using XML
  - Mobile Apps interact with Web Applications by exchanging data using JSON

Jackson

- Like JAXB, Jackson is the most often used Framework to process JSON. It helps us to convert Java Object to JSON & vice-versa

- The Process of converting Java Object to JSON is called as "Marshalling" OR "Serialization"

- The Process of converting JSON to Java Object is called as "Unmarshalling" OR "Deserialization" is called as "Marshalling" OR "Serialization"

- The Process of converting JSON to Java Object is called as "Unmarshalling" OR "Deserialization"

- "org.codehaus.jackson.*" is the package representation of Jackson

- Glassfish Jersey uses Jackson to convert the JSON data from the request to Java Object & vice-versa

- Jackson is not the only framework to process the JSON but it's one among many.
  For Ex: GSON, Moxy, JSON-P (part of Java EE) etc.,

- However Jackson is a high-performance, efficient and hence very popular JSON processor for Java

- Along with making use of JAXB related annotations Jackson also provides its own set of annotations for mapping too

Processing JSON using Jackson

- There are 3 different ways to process JSON using Jackson
  1. Streaming API (It is analogous to SAX parser for XML)
  2. Tree Model (It is analogous to DOM parser for XML)
  3. Data Binding (It converts JSON to and from Java Bean using property accessor or using annotations)

- "org.codehaus.jackson.map.ObjectMapper" its a concrete class it helps us to serialize /Marshall and deserialize/Unmarshaller regular Java objects (POJOs or Java Beans)

- There should be ONLY ONE object of "ObjectMapper" should exists per application. In other words it should be accessed in "Singleton" way. However "ObjectMapper" is not a Singleton Class

- The most commonly used methods of the ObjectMapper are
  1. writeValue()
  2. readValue()
  3. writeValueAsString()

writeValue() Methods

- There are various Overloaded Versions writeValue() Methods and all of them are Non-Static methods
- They help us to convert Java Object to JSON
- return type of these methods is "void"
- All of these methods take Object to be converted as input argument (as a 2nd input argument)

readValue() Methods

- There are various Overloaded Versions readValue() Methods
- and all of them are Non-Static methods
- They helps us to convert JSON to Java Object
- return type of these methods is "Object of the Desired Class"
- All of these methods takes JSON (in various forms) as input argument

writeValueAsString() Method

- It's a Non-Static method
- It convert Java Object to JSON & return as "String"

Jackson Marshalling Example:-

```
ObjectMapper mapper = new ObjectMapper();
mapper.writeValue(new File("fileNM.json"),
                      <Java_Object_to be_Converted>);
```

Jackson Unmarshalling Example:-

```
ObjectMapper mapper = new ObjectMapper();
Object obj = mapper.readValue(new File("fileNM.json"),
                                      <Class_NM>.class);
```