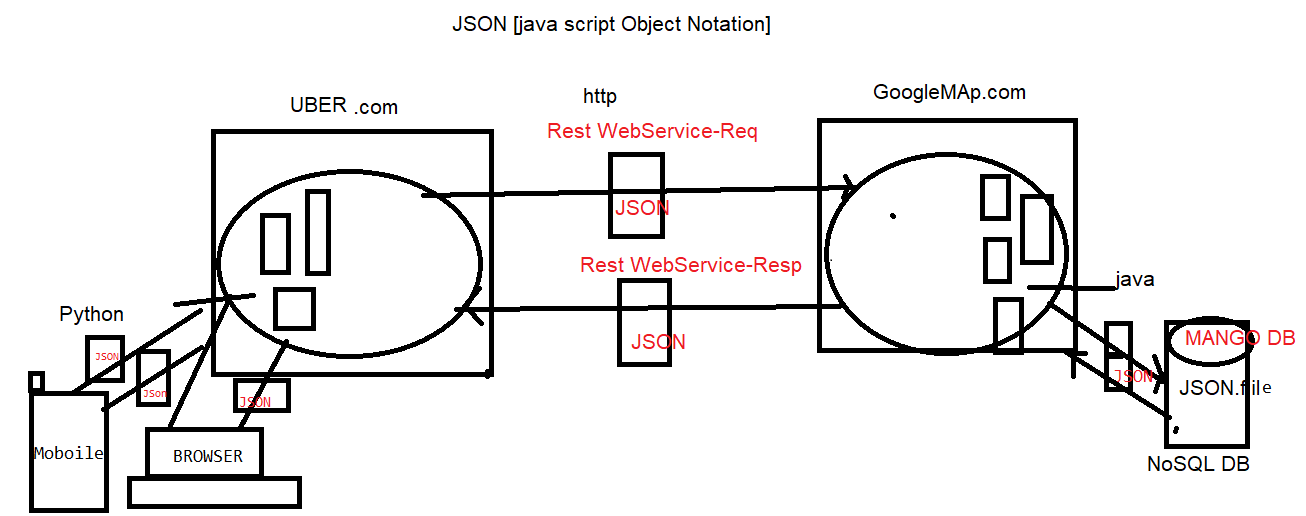
**JavaScript Object Notation [JSON]**

****

* Like XML, JSON also is a "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, as the name implies, which consists of data similar to "Object Notation of JavaScript".It's an extension of JavaScript
* 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“
* JSON syntax is derived from JavaScript object notation
* Examples of Data Formats:

**JAVA**

String str1 = "

00.12";

String str2 = "EmpID=123 | EmpNM=Deepak | EmpSal=200.12";

**XML**

<employee>

<emp-id>123</emp-id>

<emp-name>Deepak</emp-name>

<emp-salary>200.12</emp-salary>

</employee>

**JSON**

{

"EmpID":123,

"EmpNM":“Deepak",

"EmpSal":200.12

}

**JSON Syntax**

Data is in "name : value" pairs

Data is separated by "commas“

“ Curly braces" hold objects

"Square brackets" hold arrays

**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“ : “Deepak”

**JSON value**

* 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

7.an Object 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

EG:

{ "name":“Deepak", "age":35, "isEmployed":true, "girlFriend":null }

**JSON Object**

* 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)

* EG :

{

"employee":{

"name":"Praveen D",

"age":33,

"isEmployed":true,

"girlFriend":null

}

}

**JSON Array**

* 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":[ “deepak", “ram", "Malleshwar" ]

}

Example 2:-

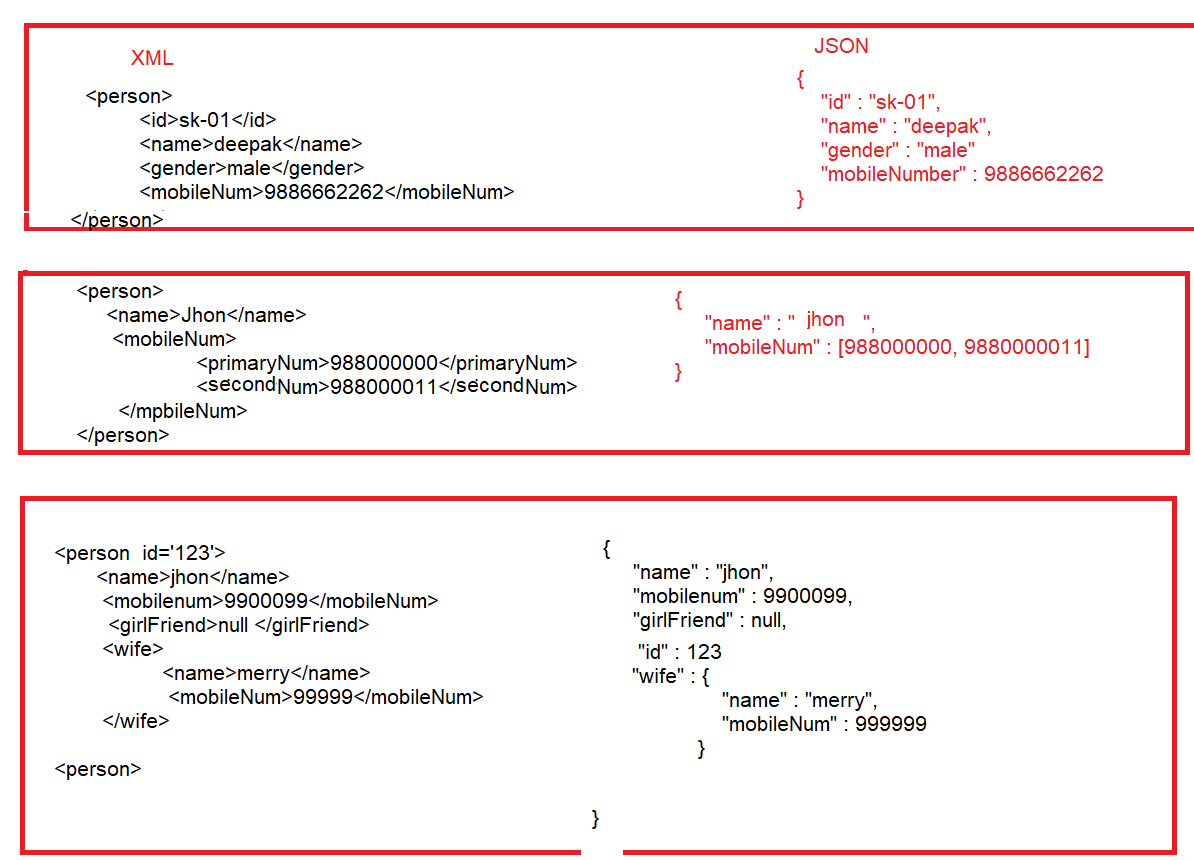
{

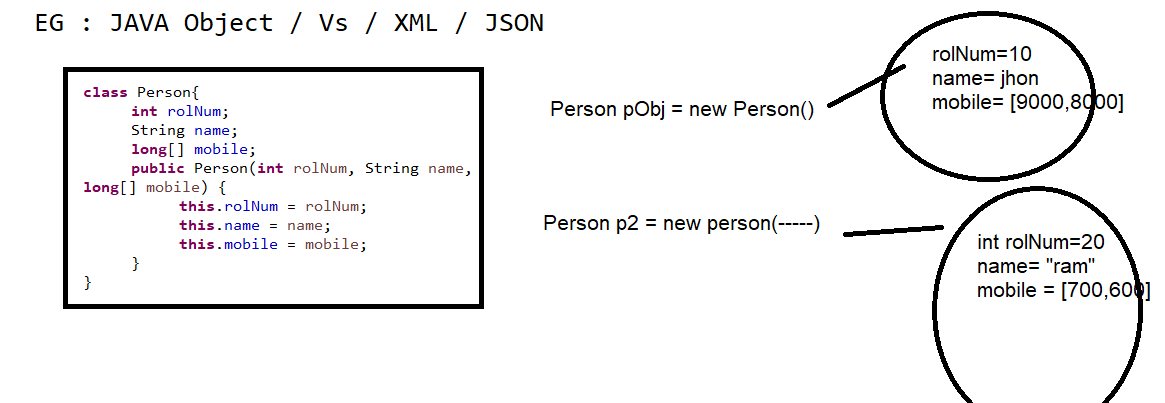
"name":“Deepak",

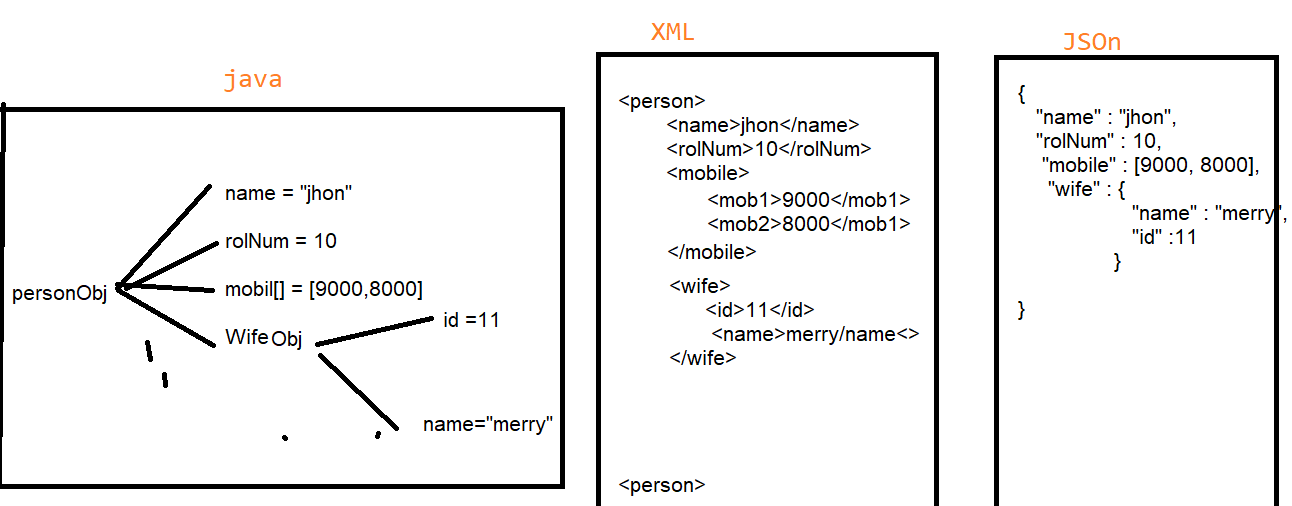
"age":33,“

cars":[ "GM", "BMW", "Audi" ]

}

****

****

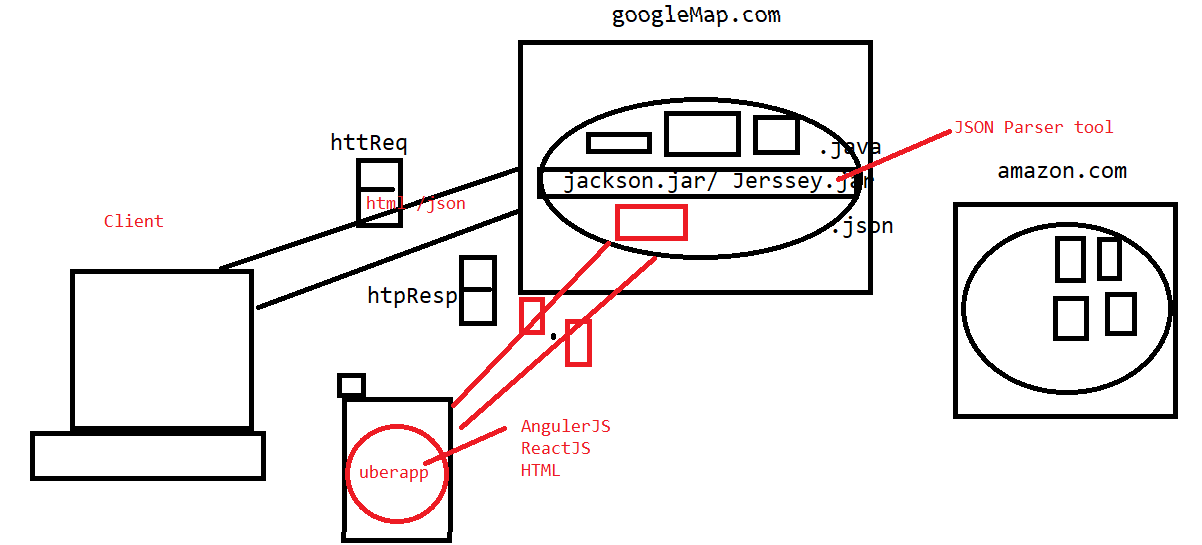
****

**JSON /XML**

|  |  |
| --- | --- |
| JSON | XML |
| JSON data has a data type , light weight | XML data is type less |
| JSON types: string, number, array, Boolean, null ,Object | All XML data should be string |
| Data is readily accessible as JSON objects | XML data needs to be parsed. |
| JSON is supported by most browsers. | Cross-browser XML parsing can be tricky |
| Retrieving value is easy | Retrieving value is difficult |
| A fully automated way of DE serialization/serialization JavaScript. | Developers have to write JavaScript code to serialize/ de-serialize from XML |
| Native support for object. | The object has to be express by conventions - mostly  missed use of attributes and elements. |
| It supports only UTF-8 encoding. | It supports various encoding. |
| It doesn't support comments. | It supports comments. |
| JSON files are easy to read as compared to XML. | XML documents are relatively more difficult to read and interpret. |
| It is less secured. | It is more secure than JSON. |

**JSON Parser**

* JAX-RS, JACSON, JERSEY is a Java API 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

****

**Write Java Parsing Program to convert JAVA To JSON**

**Or**

**Write Sterilizations java program to convert java to JSON**

**OR**

**Write** Marshalling program to convert JAVA to JSON

**Answer**

**Step 1 : add Jackson dependency in POM.xml file**

**EG :**

**<dependency>**

**<groupId>org.codehaus.jackson</groupId>**

**<artifactId>jackson-mapper-lgpl</artifactId>**

**<version>1.9.13</version>**

**</dependency>**

**Step 2 : create Java business class [Pojo Class]**

**package** pac;

**public** **class** Employe {

**int** rolNum;

String name;

**boolean** mstatus;

**int**[] moobileArr;

**public** Employe(**int** rolNum, String name, **boolean** mstatus, **int**[] moobileArr) {

**this**.rolNum = rolNum;

**this**.name = name;

**this**.mstatus = mstatus;

**this**.moobileArr = moobileArr;

}

Employe(){

}

**public** **int** getRolNum() {

**return** rolNum;

}

**public** **void** setRolNum(**int** rolNum) {

**this**.rolNum = rolNum;

}

**public** String getName() {

**return** name;

}

**public** **void** setName(String name) {

**this**.name = name;

}

**public** **boolean** isMstatus() {

**return** mstatus;

}

**public** **void** setMstatus(**boolean** mstatus) {

**this**.mstatus = mstatus;

}

**public** **int**[] getMoobileArr() {

**return** moobileArr;

}

**public** **void** setMoobileArr(**int**[] moobileArr) {

**this**.moobileArr = moobileArr;

}

}

Step 3 : create one Java Object to Pojo class

**Eg :**

**int**[] arr = {988777,98887};

Employe emp = **new** Employe(10, "yogesh", **false**, arr);

Step 4 : write parsing program using JACKSON

ObjectMapper objMapper = **new** ObjectMapper();

objMapper.writeValue(**new** File("./employe.json"), emp);

OUTPUT :



**Write Java Parsing Program to convert JSOn to JAVA**

**Or**

**Write deSerilizations java program to convert JSON to JAVA**

**OR**

**Write UN**Marshalling program to convert JSN to JAVA

Step 4 : Parsing program to convert JSON to JAVA

ObjectMapper objMapper1 = **new** ObjectMapper();

Employe e1 = objMapper.readValue(**new** File("./employe.json"), Employe.**class**);

System.***out***.println(e1.getName());