Q. 7 Explain the role of interceptors in struts.

Q. 8 What is value stack in struts? State and explain the execution flow of value stack.

Q. 9 What is ONGL? What does it consist of? How its contents can be accessed?

Q. 10 Explain the significance of struts.xml and web.xml file in Strut application.

Q. 11 Explain the application flow in MVC with respect to struts.

Q. 12 Explain the role of actions in struts.

Q. 13 Explain the role of interceptors in struts.

Q. 14 Explain the role of actions in struts.

Q. 15 What is value stack in struts? State and explain the execution flow of value stack.

Q. 16 Explain the application flow in MVC with respect to struts.

Q. 17 Explain the role of interceptors in struts.

Q. 18 What is value stack in struts? State and explain the execution flow of value stack.

Q. 19 What is ONGL? What does it consist of? How its contents can be accessed?

□□□

UNIT III

CHAPTER 7

JSON

Syllabus Topics

JSON : Overview, Syntax, DataTypes, Objects, Schema, Comparison with XML, JSON with Java

Syllabus Topic : Overview of JSON

7.1 Overview

Q. What is JSON?
Q. Explain the concept of JSON in detail.
Q. Why to use JSON? Explain it in detail.

☞ **JSON :** JSON stands for JavaScript Object Notation. Basically it is syntax for storing and exchanging data. JSON is text, written with JavaScript object notation. JSON is "self-describing" and easy to understand. It is also considered as a lightweight data-interchange format.

- JSON uses JavaScript syntax, but the JSON format is text only that's why JSON is language independent. Text can be read and used as a data format by any programming language.

☞ **Exchanging Data**

- Data exchange between a browser and a server, is always in text format. We can convert any JavaScript object into JSON, and send JSON to the server.
- We can also convert any JSON received from the server into JavaScript objects. This how simply we can work with the data as JavaScript objects, with no complications.

1. **Sending Data**

We can convert the JavaScript object into JSON, and send it to a server: Example

```
var data = { "name":"Geeta", "Age":31, "city" :"Banglore" };
```

2. **Receiving Data**

If we receive data in JSON format, we can convert it into a JavaScript object: Example

```
var data = { "name":"Geeta", "Age":31, "city" :"Banglore" };
var obj = JSON.parse(data);
document.getElementById("Mydemo").innerHTML = obj.name;
```

☞ **Why use JSON?**

1. Since the JSON format is text only, it's very easy to use for data transfer, and can be used as a data format by any programming language.
2. JSON.parse() parses a JSON string, constructing the JavaScript value or object described by the string. So, if we receive data from a server, in JSON format, we can use it like any other JavaScript object.
3. The basic use of JSON is transfer data between a server and web applications.
4. Majorly JSON is used for JavaScript based applications that includes browser extensions and websites.
5. It can be used with modern programming languages. Web services and APIs use JSON format to provide public data.
6. JSON format is used for serializing and transmitting structured data over network connection.

☞ **Example :** Fruits.json shows array of JSON elements.

```
{
  "Fruits": [
    {"name": "Mango", "rate": 250, "quantity": 12},
    {"name": "Apple", "rate": 150, "quantity": 10},
    {"name": "Banana", "rate": 40, "quantity": 12}
  ]
}
```

MyJsonProg.html : JavaScript file with JSON

```
<html>
  <head>
    <title>First JSON example</title>
    <script language = "javascript" >
      var object1 = { "name" : "Mango", "rate" : 250 };
      document.write("<h2>JSON with JavaScript example</h2>");
```

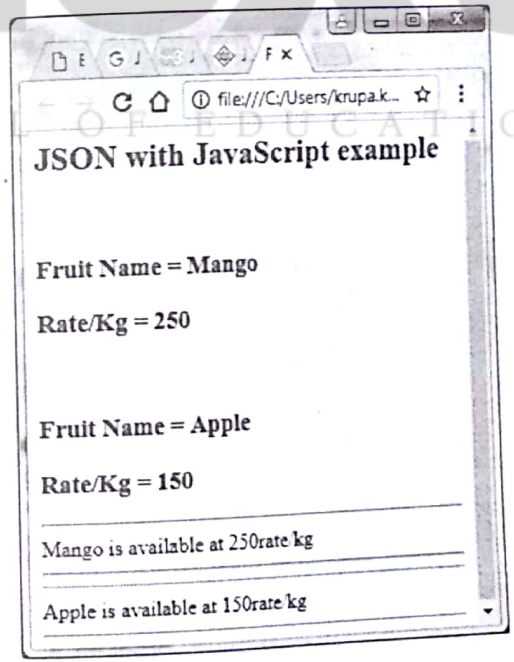


```
      document.write("<br>");
      document.write("<h3>Fruit Name = " + object1.name+"</h3>");
      document.write("<h3>Rate/Kg = " + object1.rate+"</h3>");

      var object2 = { "name" : "Apple", "rate" : 150 };
      document.write("<br>");
      document.write("<h3>Fruit Name = " + object2.name+"</h3>");
      document.write("<h3>Rate/Kg = " + object2.rate+"</h3>");
      document.write("<hr />");
      document.write(object1.name + " is available at " + object1.rate+ "rate/kg");
      document.write("<hr />");
      document.write("<hr />");
      document.write(object2.name + " is available at " + object2.rate+ "rate/kg");
      document.write("<hr />");
    </script> </head>
  <body> </body>
</html>
```

Output

This JSON program with javascript will result in following output :



Syllabus Topic : JSON Syntax

7.2 JSON Syntax

Q. Explain JSON Syntax in detail.
Q. Discuss the JSON Syntax in with suitable example.

JSON syntax is a subset of JavaScript syntax; it includes the following

- Here Data is represented in name/value pairs.
- Curly braces binds the objects and each name is followed by ':'(colon), here A name/value pair consists of a field name (in double quotes), followed by a colon, followed by a value. ***keys*** must be strings, written with double quotes, **values** must be string, number, an object (JSON object), an array, Boolean, null data types. The ***string values*** must be written with double quotes,

☞ **For Example**

```
var str={"name":"Geeta"}
var x={"salary":8900}
```

- The name/value pairs are separated by , (comma).
- Square brackets hold arrays and values are separated by ,(comma).

☞ **Array of JSON element**

Lets see one example for an array of JSON element:

```
{
  "Person": [
    { "name": "Yuvraj", "age": 25, "city": "Pune"},
    {"name": "Arvind","age": 15,"city": "Mumbai"},
    {"name": "Binita","age": 40,"city": "Banglore"}
  ]
}
```

- JSON supports varieties of data structures

1. **Collection of name/value pairs :** This Data Structure is supported by different programming languages.
2. **Ordered list of values :** It includes array, list, vector or sequence etc.

Syllabus Topic : JSON DataTypes

7.3 JSON DataTypes

Q. List and explain valid JSON data types.

In JSON, values must be one of the following data types :

1. **String :** It supports a sequence of zero or more double quoted Unicode characters. Single characters are also allowed.

☞ **Example**

```
var str={name:"Pooja"}
var str1={gender:"F"}
```

- It also supports some special characters

Special character	Meaning
"	Double quotes
\	Backslash
/	Forward slash
b	Backspace
f	Form feed
n	New line
t	tab
r	Carriage return
u	Hexadecimal digits

2. **Number :** It supports integer, floating point numbers and exponents. It doesn't allow octal, hexadecimal, NaN or infinity values.

☞ **Example**

```
var x={rate:8.9}
var y={amount:49000}
```

3. **Object (JSON object) :** This are an unordered set of key-value pair. They are always enclosed in curly braces. Here key must be unique. Objects should be used when the key names are arbitrary strings.

☞ **Example**

```
{
  "Bid": "A3425",
  "Bookname": "ADV. JAVA",
  "price": 900,
}
```

4. **Array :** it is an ordered list. It has collection of values. Its indexing may starts from 0 or 1. Each values are separated by comma(,). Array elements are enclosed in square brackets.

☞ **Example**

```
{
  "Person": [
  {"name": "Yuvraj", "age": 25, "city": "Pune"},
  {"name": "Arvind", "age": 15, "city": "Mumbai"},
  {"name": "Binita", "age": 40, "city": "Banglore" }]
}
```

5. **Boolean :** It includes true or false values.

☞ **Example**

```
var a={merit:true}
var b={merit:false}
```

6. **Null :** It is an empty type.

☞ **Example**

```
var a=null
var b={name:null}
```

7. **Whitespace :** It can be inserted between any pair of tokens. It can be added to make a code more readable.

☞ **Example**

```
var e1={name: "geeta sharma"}
var e2={name:"geetaSharma"
```

In JSON ,values cannot be function, date or anything which is undefined.

Syllabus Topic : Objects in JSON

7A JSON Objects

Q. Explain the role of JSON object in detail.
Q. Discuss the usage of JSON object with suitable example.

- JSON objects are enclosed in curly braces {}. Each element of JSON object is a **key/value pairs.**
- **Keys** must be strings, and **values** must be a valid JSON data type (string, number, object, array, boolean or null). Keys and values are separated by a **colon** (:). Each key/value pair is separated by a **comma(,).**

☞ **Example**

```
data={Bname:"Java", "price": 3400,"pages":900}
```

- We can access the object values by using dot (.) notation:

```
nm=data.Bname;
```

- We can also access the object values by using bracket ([]) notation:

```
nm=data["Bname"];
```

- We can loop through object properties by using the for-in loop.

```
<!DOCTYPE html>
<html>
<body>

<p>viewing all the elements of a JSON object</p>

<p id="demo"></p>

<script>
    var data = { "bname":"Java", "price":3400, "pages":900 };
    for (x in data)
    {
        document.getElementById("demo").innerHTML += x + "<br>";
    }
</script>

</body>
</html>
```

Output

```
viewing all the elements of a JSON object
bname
price
pages
```

- Values in a JSON object can be another JSON object. This is called nesting of JSON objects.

Example

```
data={Bname:"Java", "price": 3400,"pages":{"Part_A":300,
"Part_B":400,
"Part_C":200} }
```

- Again we can access nested JSON objects by using the dot notation or bracket notation :

```
X=data.pages.Part_A;      or
X=data.pages["Part_A"];
```

- We can modify any value in a JSON object:

```
data.pages.Part_A=600;   or
data.pages["Part_A"]=600;
```

- Use the delete keyword to delete properties from a JSON object:

```
delete data.pages.Part_B;
```

Syllabus Topic : JSON Schema

7.5 JSON Schema

Q. Explain what do you mean by JSON Schema? Explain it with suitable example.

- It is a specification for JSON based data format for defining the structure of JSON data. JSON Schema.
- It allows us to annotate and validate JSON document.

Advantages

- JSON schema describes our existing data format.

- It has complete structural validation, which is useful for automated testing.
- It has complete structural validation, which is useful for validating client-submitted data.
- It has very clear, human-readable and machine-readable documentation.

JSON Schema Validation Libraries

Various validators are currently available for various programming languages. Currently the most complete and compliant JSON Schema validator available is JSV.

Languages	Libraries
C	WJElement (LGPLv3)
.NET	Json.NET (MIT)
Java	json-schema-validator (LGPLv3)
Python	Jsonschema
PHP	php-json-schema (MIT). json-schema (Berkeley)
Ruby	autoparse (ASL 2.0); ruby-jsonschema (MIT)
JavaScript	Persevere (modified BSD or AFL 2.0); schema.js, Orderly (BSD); JSV; json-schema; Matic (MIT); Dojo;

Keywords

Let's the check various important keywords that can be used in this schema

Parameter	Keyword and Description
$schema	It is used to declare that a JSON fragment is actually a piece of JSON Schema. It also specifies the version of the JSON Schema standard.
description	A description about the schema.
title	gives a title to our schema.
properties	Defines various keys and their value types, minimum and maximum values to be used in JSON file.
type	The type keyword defines the first constraint on JSON data: it has to be a JSON Object.
minimum	It represents minimum acceptable value.
exclusiveMinimum	If "exclusiveMinimum" is present and has boolean value true, the instance is valid if it is strictly greater than the value of "minimum".
required	This stores a list of required properties.
maximum	It represents maximum acceptable value.

Parameter	Keyword and Description
exclusiveMaximum	If "exclusiveMaximum" is present and has boolean value true, the instance is valid if it is strictly lower than the value of "maximum".
pattern	If the regular expression matches the String object successfully, then it is considered as valid string object.
minLength	Here the length of a string object is defined as the minimum number of its characters.
maxLength	Here the length of a string object is defined as the maximum number of its characters.

☞ **JSON Schema Example**

Given below is a basic JSON schema, which covers a classical painting catalog description

```
{
  "$schema": "http://json-schema.org/draft-04/schema#",
  "title": "Painting",
   "description": "A painting from save child foundation",
  "type": "object",

  "properties": {

    "id": {
      "description": "The unique identifier for individual painting",
      "type": "integer"
    },

    "name": {
      "description": "Name of the painting",
      "type": "string"
    },

    "price": {
      "type": "number",
      "minimum": 0,
      "exclusiveMinimum": true
    }
```



```
  },

  "required": ["id", "name", "price"]
}
```

The above schema can be used to test the validity of the following JSON code –

```
[
  {"id": 12452, "name": "Mona Lisa", "price": 784541},
  {"id": 45473, "name": "American Gothic ", "price": 423689}
]
```

Syllabus Topic : Comparison with XML

7.6 JSON Comparison with XML

Q. Write a short note on Comparison of JSON with XML.
Q. Discuss the similarities and differences between JSON and XML.

- JSON and XML are to two most popular human readable formats and at the same time they are language independent.
- They both have support for creation, reading and decoding in real world situations. Let's compare them on following criteria :

1. **Verbose** - programmers prefer JSON over XML, as XML has more verbose. It is faster to write JSON programs.
2. **Arrays Usage** - JSON include arrays whereas XML is used to describe the structured data, which doesn't include arrays.
3. **Parsing** - JavaScript's *eval* method parses JSON. When applied to JSON, eval returns the described object.

☞ **Example** - Individual examples of XML and JSON

JSON

```
{
  "Product": "Vegetables",
  "name": "tomato",
  "price": 560
}
```

XML

```
<market>
  <product>Vegetables</product>
  <name>tomato</name>
```

```
  <price>560</price>
</market>
```

Both JSON and XML can be used to receive data from a web server. The following JSON and XML examples both defines an employees object, with an array of 3 employees :

JSON Example

```
{"customer":[
  { "firstName":"payal", "lastName":"sharma" },
  { "firstName":"shailey", "lastName":"patil" },
  { "firstName":"Peter", "lastName":"joe" }
]}
```

XML Example

```
<customers>
  <cust>
    <firstName>payal</firstName> <lastName>sharma</lastName>
  </cust>
  <cust>
    <firstName>shailey</firstName> <lastName>patil</lastName>

</cust>
  <cust>
    <firstName>Peter</firstName> <lastName>Joe</lastName>
  </cust>
</cust>
```

7.6.1 Similarities between JSON and XML

1. Both are "self describing" means human readable.
2. Both can be parsed and used by lots of programming languages.
3. Both are hierarchical indicates values are within values.
4. Both can be fetched with an XMLHttpRequest.

7.6.2 Differences between JSON and XML

1. JSON is quicker to read and write.
2. JSON can use arrays.
3. JSON doesn't use end tag.
4. JSON is shorter.
5. XML has to be parsed with an XML parser. JSON can be parsed by a standard JavaScript function.

7.6.3 Advantages of JSON over XML

1. JSON is easier to parse than XML.
2. For AJAX applications, JSON is faster and easier than XML:
3. JSON is parsed into a ready-to-use JavaScript object.

Syllabus Topic : JSON with Java

7.7 JSON with Java

Q. Explain the steps involved in executing JSON with java.
Q. Explain how to use JSON with java?

- JSON can be used with varieties of languages as discussed earlier. Here will see how to encode and decode JSON objects using Java programming language.

How to set up Environment?

- We need to first install any of the JSON modules available. For example JSON.simple is simple module used for encoding decoding, we will have to download it and install it. Add the location of **json-simple-1.1.1.jar** file to the environment variable CLASSPATH.
- The org.json.simple package contains important classes for JSON API.
- JSONValue
- JSONObject
- JSONArray
- JsonString
- JsonNumber

How to map JSON and Java entities?

JSON.simple maps entities from the right to the left while encoding and maps entities from the left side to the right side while decoding or parsing.

JSON	Java
string	java.lang.String
true|false	java.lang.Boolean
number	java.lang.Number
object	java.util.Map
array	java.util.List
null	null

Encoding JSON in Java (Example)

Let's see a simple example to encode a JSON object using Java JSONObject which is a subclass of java.util.HashMap.

```
import org.json.simple.JSONObject;
class EncodeDemo_1 {
    public static void main(String[] args)
   {
    JSONObject obj = new JSONObject();
    obj.put("name", "shreeji");
    obj.put("accno", new Integer(100));
    obj.put("balance", new Double(1000.21));
    obj.put("NRI", new Boolean(true));
    System.out.print(obj);
  }
}
```

These will results in

```
{"balance": 1000.21, "accno":100, "NRI":true, "name":"shreeji"}
```

Example

```
import org.json.simple.JSONObject;
class EncodeDemo_2 {
  public static void main(String[] args){
   JSONObject obj = new JSONObject();

    obj.put("name", "shreeji");
    obj.put("accno", new Integer(100));
    obj.put("balance", new Double(1000.21));
    obj.put("NRI", new Boolean(true));
    StringWriter wrt = new StringWriter();
    obj.writeJSONString(wrt);
    String str = wrt.toString();
    System.out.print(str);
  } }
```

These will results in

```
{"balance": 1000.21, "accno":100, "NRI":true, "name":"shreeji"}
```


Encoding JSON array using List

Let's see a simple example to encode JSON array using List in java.

```
import java.util.ArrayList;
import java.util.List;
import org.json.simple.JSONValue;
public class EncodeDemo_3
{
public static void main(String args[])
  {
  List arr = new ArrayList();
  arr.add("Jinal");
  arr.add(new Integer(25));
  arr.add(new Double(30000));
  String str = JSONValue.toJSONString(arr);
  System.out.print(str);
  }
}
```

Output

```
["jinal",25,30000.0]
```

Decoding JSON in Java

Let's see a simple example to decode JSON string in java.

```
import org.json.simple.JSONObject;
import org.json.simple.JSONValue;
public class DecodeDemo_1
{
  public static void main(String[] args)
  {
  String s="{\"name\":\"jinal\",\"salary\":30000.0,\"age\":25}";
  Object obj=JSONValue.parse(s);
  JSONObject myOBJ = (JSONObject) obj;      //typecasting to JSONObject

String name = (String) myOBJ.get("name");
```

```
    double salary = (Double) myOBJ.get("salary");
    long age = (Long) myOBJ.get("age");
    System.out.println(name+" "+salary+" "+age);
    }
}
```

Output

```
jinal  30000.0  25
```

□□□

Lab Manual

Practical 1

Aim : Develop the presentation layer of Library Management software application with suitable menus.

Ans. :

Library.java

```
import java.awt.BorderLayout;
import java.awt.EventQueue;
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.border.EmptyBorder;
import javax.swing.GroupLayout;
import javax.swing.GroupLayout.Alignment;
import javax.swing.JLabel;
import java.awt.Color;
import java.awt.Font;
import javax.swing.JButton;
import javax.swing.LayoutStyle.ComponentPlacement;
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;

public class Library extends JFrame {
  static Library frame;
  private JPanel contentPane;

  /**
  * Launch the application.
  */
  public static void main(String[] args) {
        EventQueue.invokeLater(new Runnable() {
            public void run() {
                try {
```