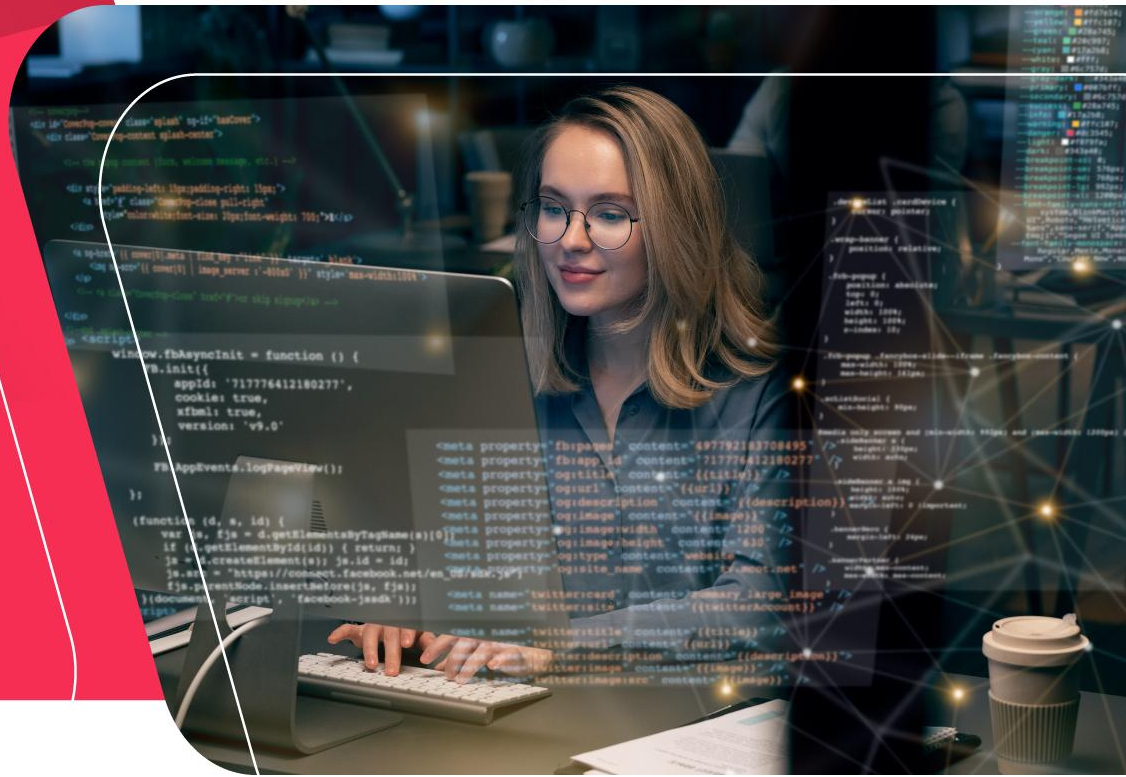
TIA-Academy

Java Bootcamp

Day 23



Technologies will be Use

- JDK 8/**11**/15
- JRE 8/**11**/15
- Intellij IDEA Community Edition
- JAVA Networking (Socket, FTP, etc)
- **JSON API**

TIA-Academy

JSON

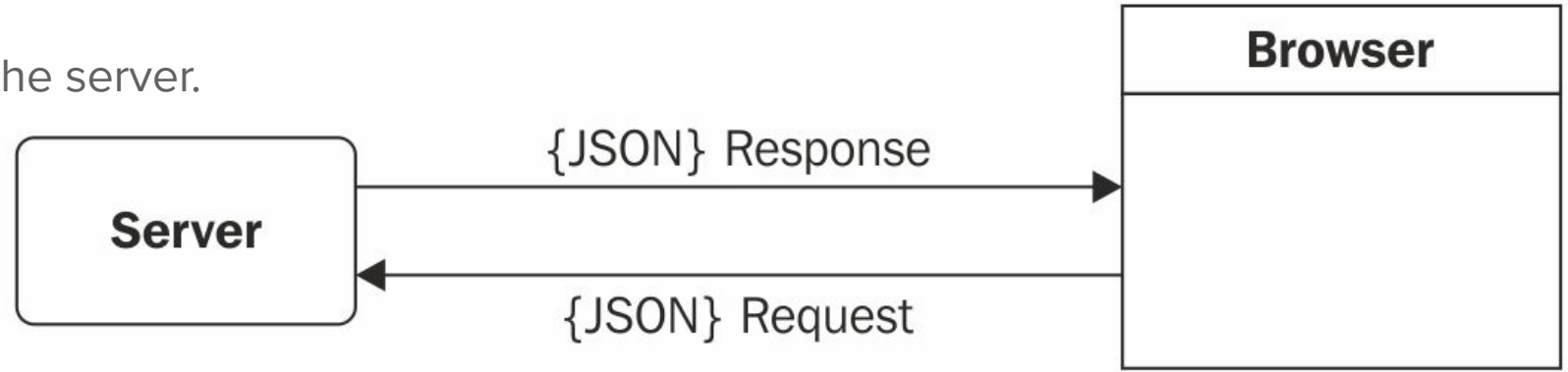
JSON - Introduction

- JSON stands for **J**ava**S**cript **O**bject **N**otation
- JSON is a lightweight data-interchange format
- JSON is "self-describing" and easy to understand
- JSON is language independent
- JSON uses JavaScript syntax, but the JSON format is text only.

Exchanging Data

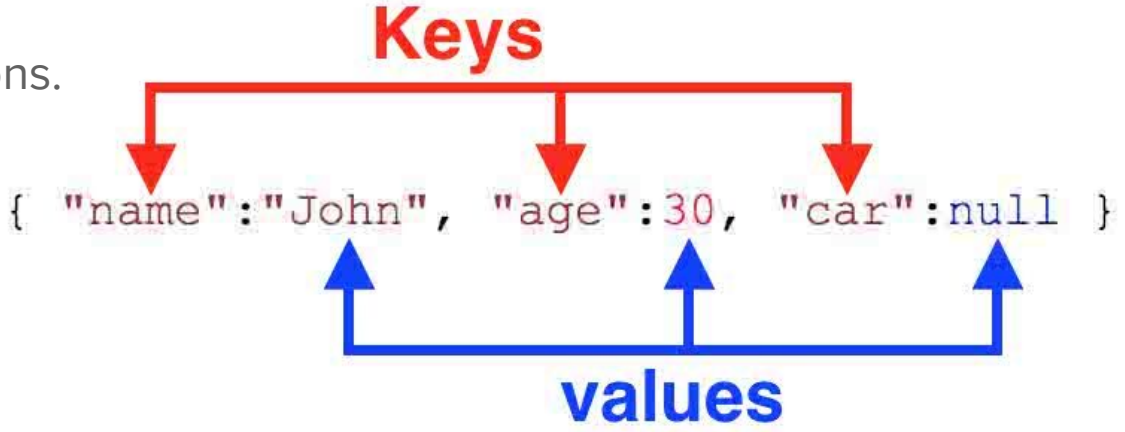
- When exchanging data between a browser and a server, **the data can only be text.**

- **JSON is text**, and we can convert any *"messages"* into JSON, and send JSON to the server.



Exchanging Data

- We can also convert any JSON received from the server into **JSON objects**.
- This way we can work with the data as JSON objects, with no complicated parsing and translations.



Why use JSON?

- Since the **JSON format is text only,** it can easily be sent to and from a server, and used as a data format by any programming language.
- Java has many library convert data, into JSON format.
- So, if you receive data from a server, in JSON format, you can decode it easily.

Why use JSON?

- 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

```
{"employees":[
    { "firstName":"John", "lastName":"Doe" },
    { "firstName":"Anna", "lastName":"Smith" },
    { "firstName":"Peter", "lastName":"Jones" }
]}
```

Why use JSON?

- XML

```
<employees>
    <employee>
        <firstName>John</firstName> <lastName>Doe</lastName>
    </employee>
    <employee>
        <firstName>Anna</firstName> <lastName>Smith</lastName>
    </employee>
    <employee>
        <firstName>Peter</firstName> <lastName>Jones</lastName>
    </employee>
</employees>
```

JSON is Like XML Because

- Both JSON and XML are **"self describing" (human readable)**
- Both JSON and XML are **hierarchical (values within values)**
- Both JSON and XML can be parsed and used by lots of programming languages
- Both JSON and XML can be fetched with an **XMLHttpRequest**

JSON is Unlike XML Because

- JSON doesn't use **end tag**
- JSON is shorter
- JSON is quicker to read and write
- JSON can use arrays
- The biggest difference is: XML has to be parsed with an XML parser.

Why JSON is Better Than XML

- XML is much more difficult to parse than JSON.
- JSON is parsed into a ready-to-use JSON object.
- For AJAX applications, JSON is faster and easier than XML:
- Using XML
    - Fetch an XML document
    - Use the XML DOM to loop through the document
    - Extract values and store in variables
- Using JSON
    - Fetch a JSON string
    - JSON.Parse the JSON string

JSON Syntax Rules

- 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

JSON Data - A Name and a Value

- JSON data is written as name/value pairs.

- A name/value pair consists of a field name (in double quotes), followed by a colon, followed by a value:

```
"name":"John"
```

- JSON names **require double quotes**. JavaScript names don't.

JSON Values

- In JSON, values must be one of the following data types:
  - a string
  - a number
  - an object (JSON object)
  - an array
  - a boolean
  - *Null*

JSON Values

- JSON values **cannot** be one of the following data types:
  - a function
  - a date
  - *undefined*

JSON Strings, Numbers, & Objects

- Strings in JSON must be written in double quotes.

```
{ "name":"John" }
```

- Numbers in JSON must be an integer or a floating point.

```
{ "age":30 }
```

JSON Strings, Numbers, & Objects

- Values in JSON can be objects.

```
{
"employee":{ "name":"John", "age":30, "city":"New York" }
}
```

- **Objects as values in JSON must follow the same rules as JSON objects.**

JSON Arrays, Booleans & null

- Values in JSON can be arrays.

```
{ "employees":[ "John", "Anna", "Peter" ] }
```

- Values in JSON can be true/false.

```
{ "sale":true }
```

- Values in JSON can be null.

```
{ "middlename":null }
```

JSON Objects

```
{ "name":"John", "age":30, "car":null }
```

- JSON objects are surrounded by **curly braces {}.**
- JSON objects are written in **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**.

Nested JSON Objects

- Values in a JSON object can be another JSON object.

```
myObj = {
    "name":"John",
    "age":30,
    "cars": {
        "car1":"Ford",
        "car2":"BMW",
        "car3":"Fiat"
    }
 }
```

JSON Arrays

```
[ "Ford", "BMW", "Fiat" ]
```

- Arrays in JSON are almost the same as arrays in JavaScript.
- In JSON, array values must be of type string, number, object, array, boolean or *null*.
- In JavaScript, array values can be all of the above, plus any other valid JavaScript expression, including functions, dates, and *undefined*.

Arrays in JSON Objects

- Arrays can be values of an object property:

```
{
"name":"John",
"age":30,
"cars":[ "Ford", "BMW", "Fiat" ]
}
```

Nested Arrays in JSON Objects

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

```
myObj = {
    "name":"John",
    "age":30,
    "cars": [
        { "name":"Ford", "models":[ "Fiesta", "Focus", "Mustang" ] },
        { "name":"BMW", "models":[ "320", "X3", "X5" ] },
        { "name":"Fiat", "models":[ "500", "Panda" ] }
    ]
}
```

JSON with JAVA

- Before you start with encoding and decoding JSON using Java, you need to install any of the JSON modules available.
- For this tutorial we have downloaded and installed JSON.simple and have added the location of **json-simple-X.X.X.jar** file to the environment variable CLASSPATH.

JSON with JAVA

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

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

Encoding JSON in Java

- Following is a simple example to encode a JSON object using Java JSONObject which is a subclass of java.util.HashMap.
- No ordering is provided. If you need the strict ordering of elements, use JSONValue.toJSONString (map) method with ordered map implementation such as java.util.LinkedHashMap.

```
import org.json.simple.JSONObject;

class JsonEncodeDemo {
   public static void main(String[] args){
      JSONObject obj = new JSONObject();
      obj.put("name", "foo");
      obj.put("num", new Integer(100));
      obj.put("balance", new Double(1000.21));
      obj.put("is_vip", new Boolean(true));
      System.out.print(obj);
   }
}
```

Encoding JSON in Java

- Following is another example that shows a JSON object streaming using Java JSONObject –

```
import org.json.simple.JSONObject;

class JsonEncodeDemo2 {
   public static void main(String[] args){
      JSONObject obj = new JSONObject();
      obj.put("name","foo");
      obj.put("num",new Integer(100));
      obj.put("balance",new Double(1000.21));
      obj.put("is_vip",new Boolean(true));
      StringWriter out = new StringWriter();
      obj.writeJSONString(out);
      String jsonText = out.toString();
      System.out.print(jsonText);
   }
}
```

Encoding JSON in Java

```
import org.json.simple.JSONObject;
class JsonEncodeDemo2 {
    public static void main(String[] args){

        //{ "employee":{ "name":"John", "age":30, "city":"New York" } }
        JSONObject objInner = new JSONObject();

        objInner.put("name", "foo");
        objInner.put("age", 30);
        objInner.put("city", "New York");

        JSONObject obj = new JSONObject();

        obj.put("empoyee", objInner);

        System.out.println(obj);
    }
}
```

Encoding JSON in Java

```
import org.json.simple.JSONArray;
import org.json.simple.JSONObject;
class JsonEncodeDemo3 {
    public static void main(String[] args){
        /* {"employees":[ "John", "Anna", "Peter" ]} */

        JSONArray arr = new JSONArray();
        arr.add("John");
        arr.add("Anna");
        arr.add("Peter");

        JSONObject obj = new JSONObject();

        obj.put("empoyee", arr);

        System.out.println(obj);
    }
}
```

Decoding JSON in Java

```
import org.json.simple.JSONObject;
import org.json.simple.parser.JSONParser;
import org.json.simple.parser.ParseException;
public class JsonDecodeDemo {
    public static void main(String args[]) throws ParseException {
        String str = "{\n" +
                "            \"balance\": 1000.21,\n" +
                "                \"is_vip\": false,\n" +
                "                \"num\": 100,\n" +
                "                \"name\": \"foo\"\n" +
                "        }";

        JSONParser parser = new JSONParser();
        Object jsonObj = parser.parse(str);
        JSONObject jsonObject = (JSONObject) jsonObj;

        String name = (String) jsonObject.get("name");
        System.out.println("Name = " + name);
        long num = (Long) jsonObject.get("num");
        System.out.println("Num = " + num);
    }
}
```

Decoding JSON in Java

```
import org.json.simple.JSONObject;
import org.json.simple.parser.JSONParser;
import org.json.simple.parser.ParseException;
public class JsonDecodeDemo2 {
    public static void main(String args[]) throws ParseException {
        // Key nya "employee", Valuenya Object
        String str = "{\n" +
                "            \"employee\": {\n" +
                "                \"name\": \"John\", \n" +
                "                \"age\": 30,\n" +
                "                \"city\": \"New York\"\n" +
                "            }\n" +
                "        }";

        JSONParser parser = new JSONParser();
        Object jsonObj = parser.parse(str);
        JSONObject jsonObject = (JSONObject) jsonObj;
        Object obj = jsonObject.get("employee");
        JSONObject empObj = (JSONObject) obj;

        String name = (String) empObj.get("name");
        System.out.println("Name = " + name);
        long age = (Long) empObj.get("age");
        System.out.println("Age = " + age);
    }
}
```

TIA-Academy

ASSIGNMENT 01

TIA-Academy

Java Command-Line Arguments

Java Command-Line Arguments

- In this tutorial, we will learn about the Java command-line arguments with the help of examples.
- The **command-line arguments** in Java allow us to pass arguments during the execution of the program.
- As the name suggests arguments are passed through the command line.

Example: Command-Line Arguments

```
class Main {
  public static void main(String[] args) {
    System.out.println("Command-Line arguments are");

    // loop through all arguments
    for(String str: args) {
      System.out.println(str);
    }
  }
}
```

Example: Command-Line Arguments

- Let's try to run this program using the command line.
- 1. To compile the code

```
javac Main.java
```

- 2. To run the code

```
java Main
```

Example: Command-Line Arguments

- Now suppose we want to pass some arguments while running the program, we can pass the arguments after the class name. For example,

```
java Main apple ball cat
```

- Here apple, ball, and cat are arguments passed to the program through the command line. Now, we will get the following output.

```
Command-Line arguments are
Apple
Ball
Cat
```

Example: Command-Line Arguments

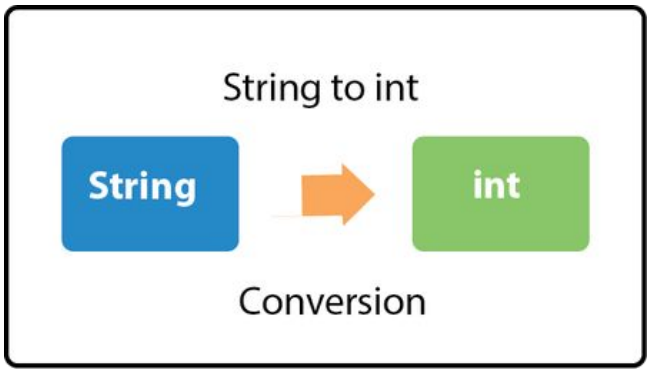
- In the above program, the main() method includes an array of strings named args as its parameter.

```
public static void main(String[] args) {...}
```

- The String array stores all the arguments passed through the command line.
- **Note:** Arguments are always stored as strings and always separated by **white-space.**

Passing Numeric Command-Line Arguments

- The main() method of every Java program **only accepts string arguments.**
- Hence it is **not possible to pass numeric arguments** through the command line.
- However, **we can later convert string arguments into numeric values**.



Example: Numeric Command-Line Arguments

- Let's try to run the program through the command line:

```
// compile the code
javac Main.java

// run the code
java Main 11 23
```

```
class Main {
  public static void main(String[] args) {

    for(String str: args) {
      // convert into integer type
    int argument = Integer.parseInt(str);
    System.out.println("Argument in integer form: " + argument);
    }

  }
}
```

Passing Numeric Command-Line Arguments

- Here 11 and 23 are command-line arguments. Now, we will get the following output.

```
Arguments in integer form
11
23
```

- In the above example, notice the line

```
int argument = Intege.parseInt(str);
```

Passing Numeric Command-Line Arguments

- Here, the parseInt() method of the Integer class converts the string argument into an integer.
- Similarly, we can use the parseDouble() and parseFloat() method to convert the string into double and float respectively.
- **Note:** If the arguments cannot be converted into the specified numeric value then an exception named NumberFormatException occurs.

TIA-Academy

ASSIGNMENT 02

TIA-Academy

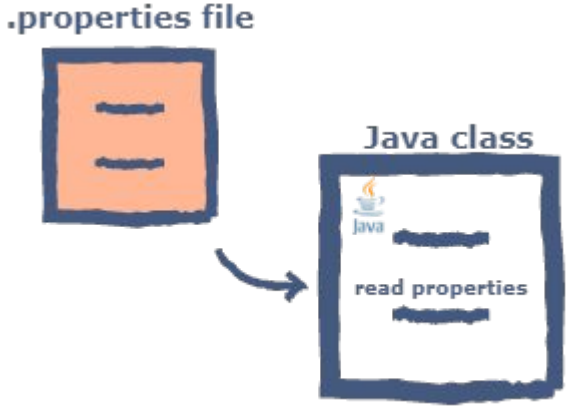
Java Properties File

Properties class in Java

- The **properties** object contains key and value pair both as a string. The java.util.Properties class is the subclass of Hashtable.
- It can be used to get property value based on the property key. The Properties class provides methods to get data from the properties file and store data into the properties file. Moreover, it can be used to get the properties of a system.

An Advantage of the properties file

- **Recompilation is not required if the information is changed from a properties file:** If any information is changed from the properties file, you don't need to recompile the java class. It is used to store information which is to be changed frequently.



Constructors of Properties class

Method	Description
Properties()	It creates an empty property list with no default values.
Properties(Properties defaults)	It creates an empty property list with the specified defaults.

Methods of Properties class

- The commonly used methods of Properties class are given below.

Method	Description
public void load(Reader r)	It loads data from the Reader object.
public void load(InputStream is)	It loads data from the InputStream object
public void loadFromXML(InputStream in)	It is used to load all of the properties represented by the XML document on the specified input stream into this properties table.
public String getProperty(String key)	It returns value based on the key.
public String getProperty(String key, String defaultValue)	It searches for the property with the specified key.
public void setProperty(String key, String value)	It calls the put method of Hashtable.
public void list(PrintStream out)	It is used to print the property list out to the specified output stream.
public void list(PrintWriter out))	It is used to print the property list out to the specified output stream.
public Enumeration<?> propertyNames())	It returns an enumeration of all the keys from the property list.
public Set<String> stringPropertyNames()	It returns a set of keys in from property list where the key and its corresponding value are strings.
public void store(Writer w, String comment)	It writes the properties in the writer object.
public void store(OutputStream os, String comment)	It writes the properties in the OutputStream object.
public void storeToXML(OutputStream os, String comment)	It writes the properties in the writer object for generating XML document.
public void storeToXML(Writer w, String comment, String encoding)	It writes the properties in the writer object for generating XML document with the specified encoding.

Example of Properties class to get information from the properties file

- To get information from the properties file, create the properties file first.

File ftp.properties

```
ftpuser=system

ftppassword=password
```

- Now, let's create the java class to read the data from the properties file.

Encoding JSON in Java

- Output:

  system

  Password

- Now if you change the value of the properties file, you **don't need to recompile the java class**.
- That means no maintenance problem.

```
import java.util.*;
import java.io.*;
public class Test {
public static void main(String[] args)throws
Exception{
    FileReader reader=new
FileReader("db.properties");

    Properties p=new Properties();
    p.load(reader);

    System.out.println(p.getProperty("user"));
    System.out.println(p.getProperty("password"));
}
}
```

Example of Properties class to get all the system properties

- By System.getProperties() method we can get all the properties of the system.
- Let's create the class that gets information from the system properties.

Example of Properties class to get all the system properties

- Output:

```
java.runtime.name = Java(TM) SE Runtime
Environment
sun.boot.library.path = C:\Program
Files\Java\jdk1.7.0_01\jre\bin
java.vm.version = 21.1-b02
java.vm.vendor = Oracle Corporation
java.vendor.url = http://java.oracle.com/
path.separator = ;
java.vm.name = Java HotSpot(TM) Client VM
file.encoding.pkg = sun.io
user.country = US
user.script =
sun.java.launcher = SUN_STANDARD
.....
```

```
import java.util.*;
import java.io.*;
public class Test {
public static void main(String[] args)throws Exception{

Properties p=System.getProperties();
Set set=p.entrySet();

Iterator itr=set.iterator();
while(itr.hasNext()){
Map.Entry entry=(Map.Entry)itr.next();
System.out.println(entry.getKey()+" = "+entry.getValue());
}

}
}
```

Example of Properties class to create the properties file

- Now let's write the code to create the properties file.
- Let's see the generated properties file.
- **info.properties**

```
email=sonoojaiswal@javatpoint.com
name=Sonoo Jaiswal
```

```
import java.util.*;
import java.io.*;
public class Test {
public static void main(String[] args)throws
Exception{

Properties p=new Properties();
p.setProperty("name","Sonoo Jaiswal");
p.setProperty("email","sonoojaiswal@javatpoint.com");

p.store(new FileWriter("info.properties"),"Javatpoint
Properties Example");

}
}
```

TIA-Academy

ASSIGNMENT 03

TIA-Academy

ASSIGNMENT 04
(HOME ASSIGNMENT)

TIA-Academy

Thank You