Java Programming

Arthur Hoskey, Ph.D.
Farmingdale State College
Computer Systems Department

- Go over important dates

Important Dates

- JavaScript Object Notation (JSON)
- Importing GSON library
- Using the GSON library

Today's Lecture

- JavaScript Object Notation (JSON)

- Format used to transfer data.

- Line:
www.json.org

JSON

Car class storing only the year…

Start of object

Colon separates name and value

```
{
        "year" : 2015
}
```

Variable Name

Value

End of object

Stores member variable names and their values together (name-value pairs)

- Variable names: Must be surrounded by double quotes.
- Variable values:
  - String value – Must be surrounded by double quotes
  - Any other value - NO double quotes

JSON

Car class storing year, speed, and color…

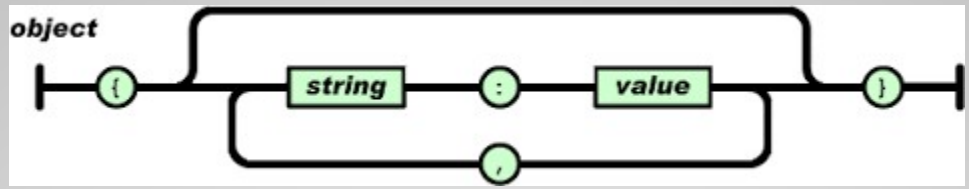
```
{
        "year" : 2015  ,
        "speed" : 20   ,
        "color" : "red"
}
```

Use commas to separate name-value pairs

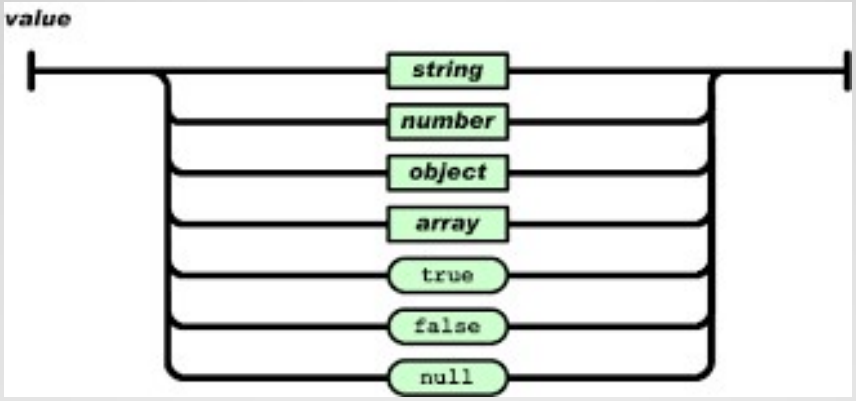
This object contains multiple member variables and their values

JSON

- www.json.org contains diagrams showing the correct sequence of characters for various types of data.
- Here is a railroad diagram for a JSON object:



- Here is a railroad diagram for a JSON value:



JSON Railroad Diagrams

JSON Library

- Can use a JSON library to create and consume JSON.
- A JSON library is not part of the JDK.
- GSON – Google's JSON library.
- The following slides detail how to import the GSON library into an Apache NetBeans project that uses Maven Java Application Project.

- The following discusses importing the GSON library using Maven.
- This will only work if you create a Java with Maven type project in Apache NetBeans.
- You must first get the GSON library dependency info (groupId, artifactId, version).

Find GSON Library Dependency Info

1. Find the GSON library in the Maven repository. Here is a link: https://mvnrepository.com/artifact/com.google.code.gson/gson
2. Click the link for the version you want to use.
3. Choose the Maven tab to get the dependency info.

Import GSON Library Using Maven

Add Dependency in Apache NetBeans

1. In the Projects window right-click the Dependencies folder. Choose Add Dependency from the context menu.

2. Fill in the groupId, artifactId, and version. Click Add when done. The GSON library should now appear under the Dependencies folder for the project.

Import GSON Library Using Maven

Employee class

- No mapping attributes

```
public class Employee {
   private String name;
   private int id;
}
```

If member variables do not have mapping attributes then the class variable names must match the JSON variable names

Required JSON

```
{
  "name": "Jane Smith",
  "id": 200
}
```

Not using mapping so JSON variable names MUST be exactly the same as the class variable names

Class Setup for JSON

Employee class

- Mapping attributes
- @SerializedName – Attribute that maps class variables to JSON variables

```
Import
public class Employee {
   @SerializedName("namexyz")
   private String name;

   @SerializedName("id")
   private int id;
}
```

Class member variables are mapped to JSON variables. The class variable names do NOT have to match the JSON variable names

Required JSON

```
{
  "namexyz": "Jane Smith",
  "id": 200
}
```

Class Setup for JSON

- You must create some GSON related variables to parse or create JSON.

```
GsonBuilder builder = new GsonBuilder();
builder.setPrettyPrinting();
Gson gson = builder.create();
```

The gson variable can now be used to create or parse JSON

Note: Pretty printing means it will cause the JSON to appear on different lines and be properly indented.

Setup GSON Variables

- **toJson** – This method generates a JSON string given an object.

JSON OUTPUT

```
{
  "name": "Jane Smith",
  "id": 200
}
```

```
Employee e = new Employee();
e.setName("Jane Smith");
e.setId(200);

GsonBuilder builder = new GsonBuilder();
builder.setPrettyPrinting();
Gson gson = builder.create();

String jsonString = gson.toJson(e);
```

Setup GSON

Pass the object into the toJson method. This method returns the JSON string for the object.

Create JSON String from Object

- **fromJSON – String version.** Takes a JSON string and sets an object's member variable values.
- After the code below runs e2 will have "Jane Smith" and 200 in its member variables.

```
String jsonString = "{ \"name\": \"Jane Smith\",  \"id\": 200 }";
OR
String jsonString = Read it from a file (you will most likely do this)

Employee e2 = gson.fromJson(jsonString, Employee.class);
```

JSON string contain values to set member variables to

Employee.class has data about the member variable names and types

Create Object from JSON String

- **fromJSON – File version.** Takes a FileReader as its first parameter.
- Open a FileReader and pass it in to fromJson.

Employee.json File

```
{
  "name": "Jane Smith",
  "id": 200
}
```

Input File

Open File

```
FileReader fr = new FileReader("Employee.json");
Employee e2 = gson.fromJson(fr, Employee.class);
```

Pass in FileReader

```
String name = e2.getName();
int id = e2.getId();
// Next, you copy name and id into member variables
```

Get the data that you need from the instance that fromJson created.

Create Object from JSON File

- **toJson** – Generates a JSON string from an object.
- The example below writes the JSON string to a file.

EmpOut.json

```
{
  "name": "Jane Smith",
  "id": 200
}
```

```
Employee e = new Employee();
e.setName("Jane Smith");
e.setId(200);

GsonBuilder builder = new GsonBuilder();
builder.setPrettyPrinting();
Gson gson = builder.create();

String jsonString = gson.toJson(e);

PrintStream ps = new PrintStream("EmpOut.json");
ps.println(jsonString);
```

Setup GSON

Pass the object into the toJson method. This method returns the JSON string for the object.

Write JSON to file

Write JSON to File

- Reading an array from a file is similar to reading a class from a file (use array data type instead).

```
FileReader fr = new FileReader("EmployeeArray.json");
Employee[] ea = gson.fromJson(fr, Employee[].class);
```

Employee[].class tells GSON that it is reading an array of Employee

EmpArray.json

```
[
  {
    "name": "Jane Smith",
    "id": 200
  },
  {
    "name": "Jose Diaz",
    "id": 300
   }
]
```

Read JSON Array From File

- The example below writes a JSON array to a file.

```
Employee[] ea = new Employee[2];
// Code to fill array goes here...

GsonBuilder builder = new GsonBuilder();
builder.setPrettyPrinting();
Gson gson = builder.create();

String jsonString = gson.toJson(ea);

PrintStream ps = new PrintStream("EmpArrayOut.json");
ps.println(jsonString);
```

EmpArray.json

```
[
  {
    "name": "Jane Smith",
    "id": 200
  },
  {
    "name": "Jose Diaz",
    "id": 300
  }
]
```

Write JSON Array to File

- Take Attendance!

Attendance