Java Programming

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

Today's Lecture

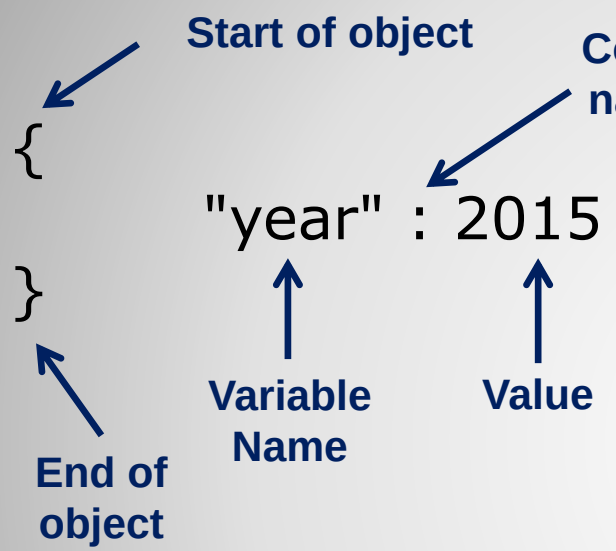
- JavaScript Object Notation (JSON)
- Importing GSON library
- Using the GSON library

- JavaScript Object Notation (JSON)
- Format used to transfer data.
- Line:

www.json.org

JSON

Car class storing only the year…



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

One JSON Object

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

Array of car JSON objects…

```
[
  {
    "year" : 2015,
    "speed" : 20,
    "color" : "red"
  },
  {
    "year" : 2019,
    "speed" : 30,
    "color" : "green"
  },
  {
    "year" : 2022  ,
    "speed" : 40   ,
    "color" : "blue"
   }
]
```

Start of JSON array

Use commas to separate each object of the array

End of JSON array

Use [ ] to define an array JSON objects

Array of JSON Objects

JSON Library

- Use a JSON library to create and consume JSON (much easier than doing it yourself).
- 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 IntelliJ project that uses Maven Java Application Project.

Steps to Import GSON Library into IntelliJ

1. Add the GSON Maven dependency to the project.
2. If you are importing into an IntelliJ JavaFX project (GUI), you must also modify the module-info.java file.

Import GSON Library Using Maven

Add GSON Maven Dependency

- This will only work if you create a Java with Maven type project in IntelliJ.
- 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 (select the latest version).
3. Choose the Maven tab to get the dependency info.

```
<dependency>
    <groupId>com.google.code.gson</groupId>
    <artifactId>gson</artifactId>
    <version>2.10.1</version>
</dependency>
```

1. Add GSON Maven Dependency

Add Dependency in IntelliJ

1. Open pom.xml file.
2. Inside pom.xml, find the <dependencies> node.
3. Add the GSON dependency as a child of <dependencies>.

For example:

```
<dependencies>
    <dependency>
        <groupId>com.google.code.gson</groupId>
        <artifactId>gson</artifactId>
        <version>2.10.1</version>
    </dependency>

    <!-- Other dependencies here -->
</dependencies>
```

Note: If there is no <dependencies> node then add one.

1. Add GSON Maven Dependency

IntelliJ JavaFX Project – module-info.java

- For IntelliJ JavaFX projects you must also modify the module-info.java file to use the GSON library.

1. In the Projects window in IntelliJ navigate to src/main/Java and open the module-info.java file.
2. Add the following requires statement for the GSON package:

requires com.google.gson; // Add near top

3. Add the GSON package to the opens statement:

opens <your project> to javafx.fxml; // Original

opens <your project> to javafx.fxml, com.google.gson; // New

If you get the following error: Module not found: com.google.gson

Go to Maven tab (on right). Press Reload All Maven Projects in the toolbar.

First button is for reload

2. Modify module-info.java (only for FXML Projects)

Employee class

- No mapping attributes (not the best way to do it)

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

   // Get/set methods here…
}
```

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

- Uses mapping attributes (associates Java variables with JSON variables)
- @SerializedName – Annotation that maps class variables to JSON variables

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

   @SerializedName("id")
   private int id;

   // Get/set methods here…
}
```

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

Map the name variable to the namexyz JSON variable

Required JSON

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

Class Setup for JSON

- **fromJSON – File version.** fromJson takes a FileReader as a parameter.

Employee.json (Input File)

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

Note: You will need to add try/catch code because a file is being opened

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

Setup GSON

Open File

```
FileReader fr = new FileReader("Employee.json");
```

Pass in FileReader

```
Employee e = gson.fromJson(fr, Employee.class);
```

Pass in the data type of the JSON being read

```
String name = e.getName();
int id = e.getId();
```

e has been filled with data from the JSON file, now use it

Read JSON From File

- **toJson** – 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();
Gson gson = builder.create();

String jsonString = gson.toJson(e);

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

Setup GSON

Pass the object e into the toJson method. This method returns a JSON string for the object (it returns a normal string that is formatted as JSON).

Write jsonString to EmpOut.json file

Write Object to a JSON File

Read JSON Array From 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);
```

EmpArray.json

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

Employee[].class tells GSON that it is reading an array of Employee (not one single employee)

- 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

Read Into an ArrayList Collection

- If you need to read in a **collection of reference type** you should use the following:

```
ArrayList<Employee> list = null;

FileReader fr = new FileReader("EmpList.json");
list = gson.fromJson(fr, new TypeToken<ArrayList<Employee>>(){}.getType());
```

Use this code that uses TypeToken instead of the .class syntax

Read JSON Array From a File Into an ArrayList

- End of Slides

End of Slides