CMPS 251

Read/Write Files

Dr. Abdelkarim Erradi

Computer & Engineering Science Dept.

QU

Outline

- **Read / Write Text File**
- **Read / Write JSON File**

Read / Write Text File

Back

Overview

- Data stored in a program variables is lost when the program ends
- To store data between program runs, we use files
- Java has MANY ways to read and write files
  - We will focus on the most commonly used ways

Use Java 8 java.nio.file package

- **Representing file paths**

  Paths.get
- **Reading files**

  Files.lines
- **Writing files**

  Files.write
- **Exploring folders**

  Files.list, Files.walk, Files.find

Paths

- **Paths** class provides a way to represent a file path and get path info
- **Get Path with Paths.get**
  - `Path p1 = Paths.get("some-file");`
  - `Path p2 = Paths.get("/usr/local/gosling/some-file");`
  - `Path p3 = Paths.get("C:\\Users\\ae\\some-file");`
    - Notice the double backslashes because backslash is used to escape next char in Java strings.
- **Paths have convenient methods**
  - toAbsolutePath, getFileName, getParent, getRoot …

Example

```
public static void main(String[] args) {
 Path path =
    Paths.get("data/countries.json").toAbsolutePath();
 System.out.printf("Absolute Path: %s%n", path);
 System.out.printf("getFileName: %s%n",
                           path.getFileName());
 System.out.printf("getParent: %s%n",
                           path.getParent());
 System.out.printf("getRoot: %s%n", path.getRoot());
}
```

```
Absolute Path: D:\cmps251\cmps251-content\Examples\10.FileIO\data\countries.json
getFileName: countries.json
getParent: D:\cmps251\cmps251-content\Examples\10.FileIO\data
getRoot: D:\
```

Read File Content

- **You can read all lines into Stream in 1 method call**

```
Stream<String> lines = Files.lines(somePath);
```

- **Quick example**
  - Get Middle East countries from countries.txt file and save them to me-countries.txt file

```
String inputFileName = "data/countries.txt";
String outputFileName = "data/me-countries.txt";
List<String> countries =
  Files.lines(Paths.get(inputFileName))
       .filter(c -> c.contains("Middle East"))
       .map(c -> c.split(";")[0])
       .sorted()
       .collect(Collectors.toList());

System.out.println(countries);
Files.write(Paths.get(outputFileName), countries);
```

Benefits of Files.lines

- **`Files.lines` return a Stream**
  - Much faster + memory savings
  - Does not store entire file contents in one huge list, but processes each line as you go along
  - You can stop partway through, and rest of file is never processed (due to lazy evaluation of Streams). E.g., using **.findFirst()**
- Many convenient filtering and transformation methods
  - You can chain these method calls together

Files.write

- **You can write all lines in one method call**
  - `List<String> lines = …;`
  - `Files.write(somePath, lines);`
- **You can write all bytes in one method call**
  - `byte[] fileArray = …;`
  - `Files.write(somePath, fileArray);`
- **OpenOption**
  - Both methods above optionally take an OpenOption to specify whether to create file if it doesn't exist, whether to append.
  - Default behavior is to create file if not there and to overwrite if it exists

Example

- Write a list of Strings to a file

```
Path path = Paths.get("data/testFile.txt");
List<String> lines =
  List.of("Line One", "Line Two", "Final Line");
Files.write(path, lines);
```

Get files of a folder

- **Get all files in a folder:** `Files.list`

```
Files.list(Paths.get(folder))
  .forEach(System.out::println);
```

Writing Formatted Text to a File

- Class **`PrintWriter`** (from package **`java.io`**) defines methods to create and write to a text file
  - To open the file Declare a *variable* using **`PrintWriter`** constructor, pass file path as argument
  - Wrap in a **`try`** and **`catch`** blocks to handle any IOException such as a new file cannot be created
- Use **`println / printf`** method to write to the file
- Close the file when done

Writing Formatted Text to a File: Example

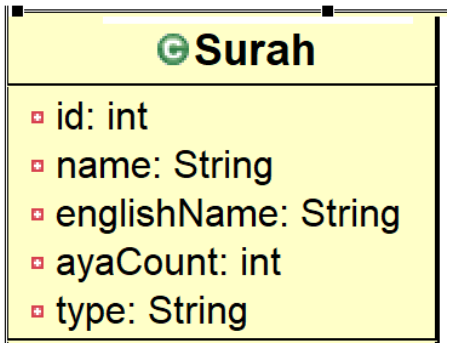
```
PrintWriter out = new
PrintWriter("data/MyFormattedFile.txt");
out.println("This is being written to a file.");


for (int i = 0; i < 10; i++) {
      out.printf("%d%n", i);
}


out.close();
```

Read / Write JSON File

JSON Data Format

- **JSON** (JavaScript Object Notation) is a very popular lightweight data format to transform an object to a **text** form to ease storing and transporting data
- **Gson** library could be used to transform an object to json or transform a json string to an object

Transform an instance of Surah class to a JSON string:




```
Gson gson = new Gson();
Surah surah = new Surah(1, "الفاتحة" ,"Al-Fatiha", 7, "Meccan");
String surahJSON = gson.toJson(surah);
```

```
{
    "id": 1,
    "name": "الفاتحة",
    "englishName": "Al-Fatiha",
    "ayaCount": 7,
    "type": "Meccan"
}
```

Read / Write JSON file

- Read a JSON file and convert its content to objects

```
Gson gson = new Gson();
String filePath = "data/surah.json";
String fileContent = Files.readString(Paths.get(filePath));
Surah[] surahs = gson.fromJson(fileContent, Surah[].class);
```

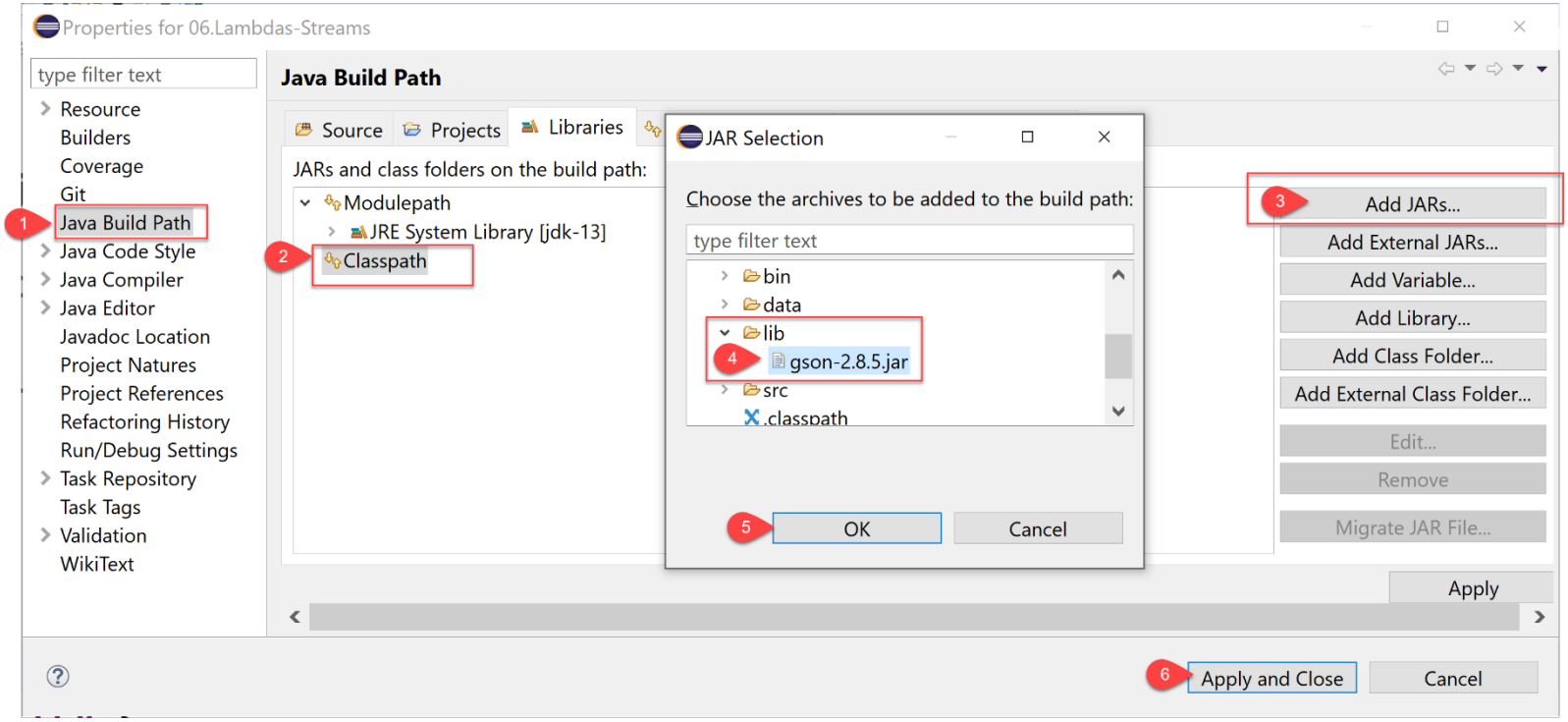
- Write objects to a JSON file

```
String surahsJSON = gson.toJson(surahs);
Files.writeString(Paths.get(filePath), surahsJSON);
```

You may use https://codebeautify.org/json-to-java-converter to generate a Java class from a json string!

Steps to use Gson library

- Create a subfolder named **lib** under your project folder
- Download **Gson** library into **lib** subfolder
  https://repo1.maven.org/maven2/com/google/code/gson/gson/2.8.5/gson-2.8.5.jar
- Write-click your project and select **`Properties…`**
- Select ***`Java Build Path`***. Click ***`Classpath`*** then click ***`Add JARS…`*** select ***`gson-2.8.5.jar`*** from your project **lib** subfolder (see image below)



Reading and writing JSON files for Class with JavaFX properties

- Use **FxGson** for classes with JavaFX properties

https://repo1.maven.org/maven2/org/hildan/fxgson/fx-gson/3.1.2/fx-gson-3.1.2.jar

```
Gson gson =
FxGson.coreBuilder().setPrettyPrinting().create();
```

- No change to the code to read/write json files

Summary

- **Use Path to refer to file location**

```
Path somePath = Paths.get("/path/to/file.txt");
```

- **Read all lines into a Stream**

```
Stream<String> lines = Files.lines(somePath);
```

  - Can now use filter, map, distinct, sorted, findFirst, etc.
  - You get benefits of lazy evaluation
  - Can output as List with `collect(Collectors.toList())`

- **Write List into a file**

```
Files.write(somePath, someList, someCharset);
```

- **Use PrintWriter for more flexible output**
- **Use Gson library for reading/writing json files**
  - Use **FxGson** for classes with JavaFX properties