Module 1-17

File Output

Java Output

Java has the ability to communicate data back to the user. Consider some of these methods:

- Using System.out.println() that sends a message to the console.
- Send a HTML view back to the user (Module 3).
- Write data to a database (Module 2).
- Transmit data to an API (Module 4).

For today, we will focus on something simpler, writing data back to a text file.

File class: create a directory.

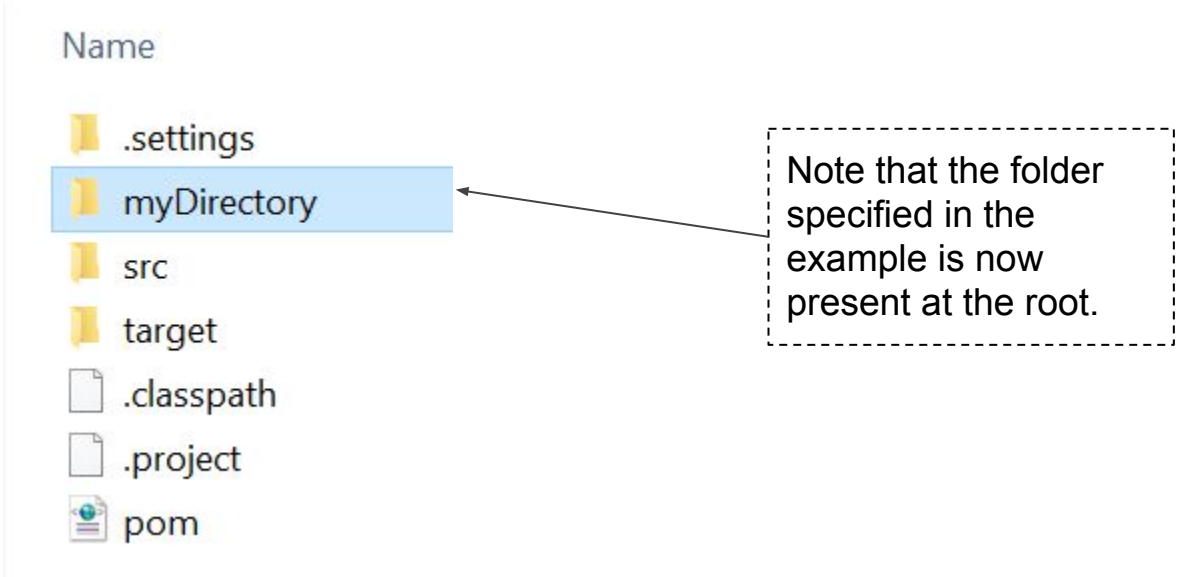
```
public static void main(String[] args) {
        File newDirectory = new File("myDirectory");

        if (newDirectory.exists()) {
                System.out.println("Sorry, " + newDirectory.getAbsolutePath() + " already exists.");
        }
        else {
                newDirectory.mkdir();
        }
}
```

We won't create a new directory if it exists.

Otherwise, the .mkdir method will create a new directory.

File class: create a directory.

Just like with reading from files, writing is done with respect to the project root.



File class: create a file.

The file constructor can take only a file name, the file will be created at the root of the project.

```
File newFile = new File("myDataFile.txt");
newFile.createNewFile();
```

The File constructor can also take in two arguments, the directory, and the file name, causing the file to be created at the specified folder.

```
File newFile = new File("myDirectory","myDataFile.txt");
newFile.createNewFile();
```

Writing to a File

Just like with reading data from a file, writing to a file involves bringing in an object of another class. In this case, we will need an instance of the PrintWriter class.

Writing a File Example

```
public static void main(String[] args) throws IOException {
        File newFile = new File("myDataFile.txt");
        String message = "Appreciate\nElevate\nParticipate";

        PrintWriter writer = new PrintWriter(newFile.getAbsoluteFile());
        writer.print(message);
        writer.flush();
        writer.close();
}
```

Create a new file object.

Create a PrintWriter object.

print the message to the buffer.

flush the buffer's content to the file.

The expected result:

- There will be a new text file in the project root.
- The file will be called myDataFile.txt
- The file will contain each of the three words in its own line.

What is a buffer?

A buffer is like a bucket where the text is initially written to. It is only after we invoke the **.flush()** method that the bucket's contents are transferred to the file. **The flush() and the close() can be performed automatically if the "try with resources" is used:**

```
public static void main(String[] args) throws IOException {
        File newFile = new File("myDataFile.txt");
        String message = "Appreciate\nElevate\nParticipate";

        try(  PrintWriter writer = new PrintWriter( newFile.getAbsoluteFile() )  ) {
                writer.print(message);
        }
}
```

Appending to a File

The previous example regenerates the file's contents from scratch every time it's run. Sometimes, a file might need to be appended to, preserving the existing data content. The PrintWriter supports two constructors:

- **PrintWriter**(**file**), where file is a file object.
- **PrinterWriter**(**new OutputStream(file,true)**)

Appending a File Example

```
public static void main(String[] args) throws IOException {
		File newFile = new File("myDataFile.txt");
		String message = "Appreciate\nElevate\nParticipate";

		PrintWriter writer = null;

		// Instantiate the writer object with append functionality.
		if (newFile.exists()) {
			writer = new PrintWriter(new FileOutputStream(newFile.getAbsoluteFile(), true));
		}

		// Instantiate the writer object without append functionality.
		else {
			writer = new PrintWriter(newFile.getAbsoluteFile());
		}

		writer.append(message);
		writer.flush();
		writer.close();

	}
}
```

The expected result is that *myDataFile.txt* will be continuously appended with the message text each time it runs.