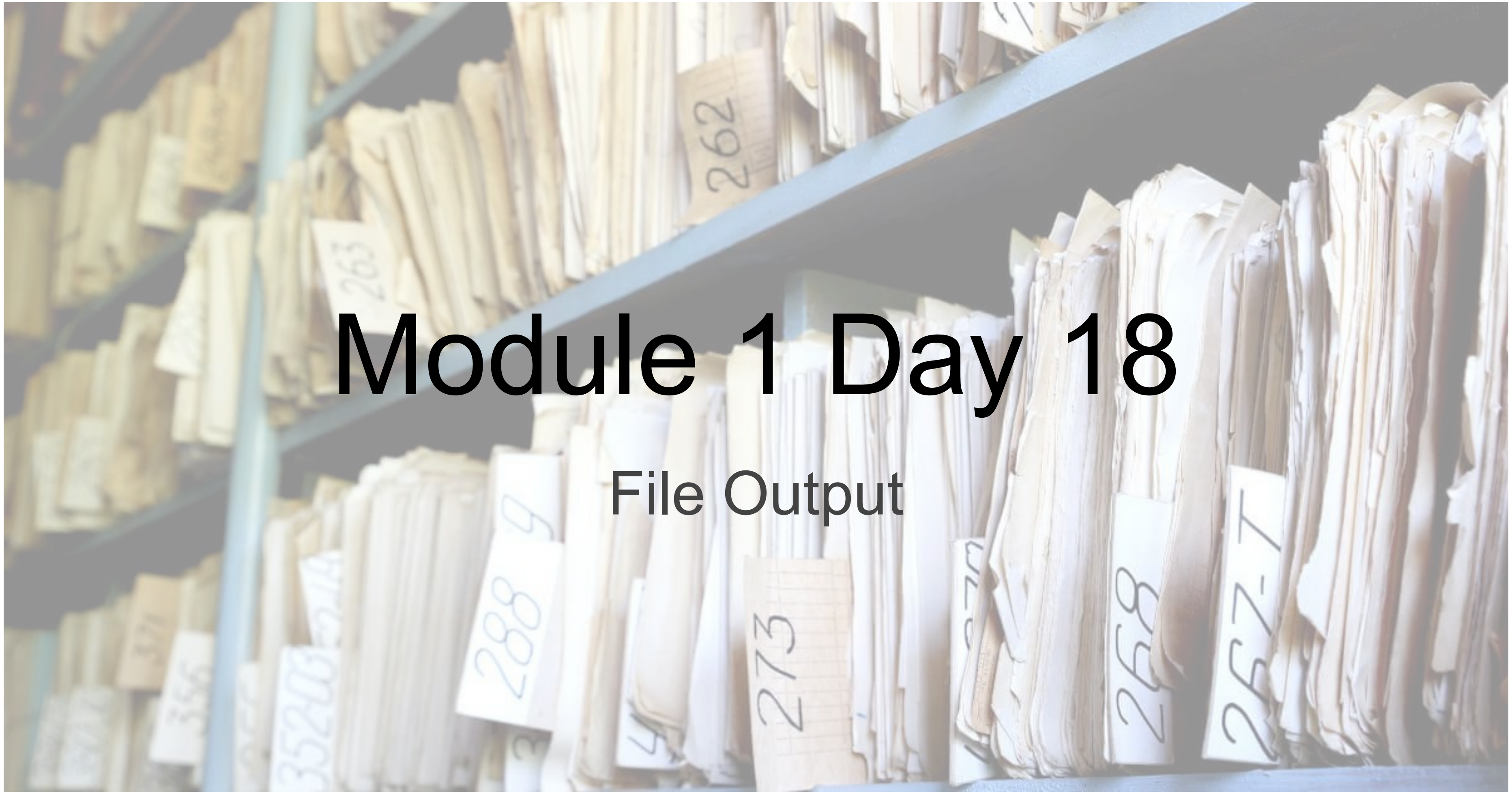


Module 1 Day 18

File Output

Module 1 Unit 18 File Output

Can you … ?

- … describe the concept of exception handling
- … implement a try/catch structure in a program
- … use and discuss the `System.IO` namespace (C#) / `java.io` library File and Directory classes
- … explain what a character stream is
- … use a try-with-resources block
- ... handle File I/O exceptions and how to recover from them
- … talk about ways that File I/O might be used on the job

Java Output

Java, like all languages, can communicate data, as output, to the user. This output can occur in various ways:

- 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 3).

Today, we will focus on writing data back to a text file.

File class: create a directory.

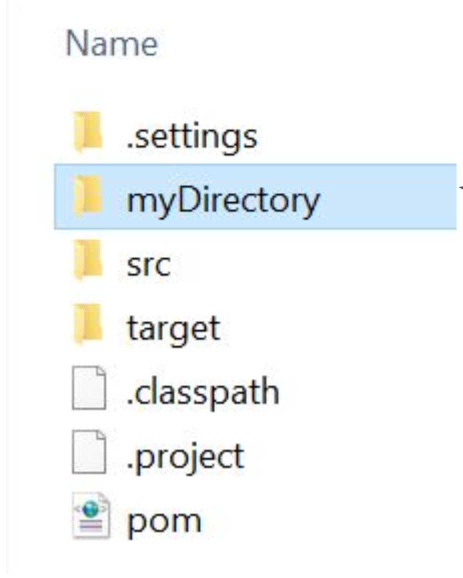
```
public static void main(String[] args) {
        File newDirectory = new File("myDirectory");

        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 relative to the project root unless an absolute path is provided for a directory.



Name

.settings
myDirectory
src
target
.classpath
.project
pom

Note that the folder specified in the example is now present at the root.

File class: create a file.

```
public static void main(String[] args) throws IOException {
        File newFile = new File("myDataFile.txt");
        newFile.createNewFile();
}
```

File class: create a file within a directory.

```
public static void main(String[] args) throws IOException {
    File newFile = new File("myDirectory","myDataFile.txt");
    newFile.createNewFile();
}
```

Writing to a File

- Just like reading data from a file using Scanner, writing to a file involves the use of an object of another class: PrintWriter.

- When more than one class is used to solve a problem, we refer to those classes as **collaborators**. In the case of writing files, the File and Printwriter classes are collaborators.

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.

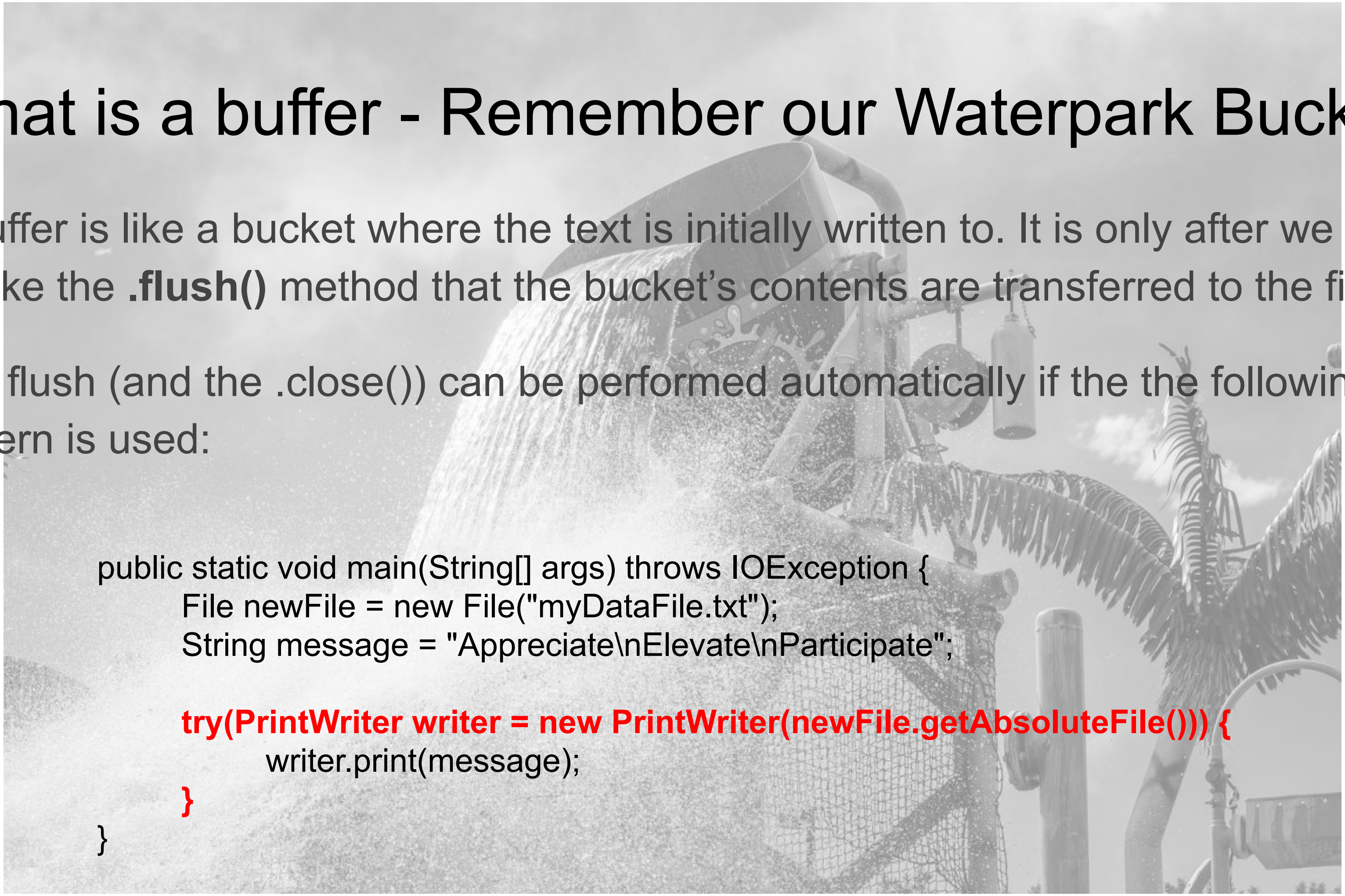
print the message to the buffer.

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 the text of ***message*** each of the three words on its own line due to the ***\n*** newline escape character..

What is a buffer - Remember our Waterpark Bucket?



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 the following pattern 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. While this is fine, at other times a file may need to have data appended to it to preserving existing data. The PrintWriter supports two constructors:

- **PrintWriter**(**file**), where file is a file object.
- **PrinterWriter**(**outputStream**, **mode**)
    - outputStream will be an instance of the OutputStream class.
    - Mode is a boolean indicating if you want to instantiate the object in append mode (true = yes).

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 to with **message** each time this code runs.*