

FILE I/O

PART 2

TODAY'S OBJECTIVES

- Investigating File and Directory metadata using the File class
- Write text data to a file
- Buffering in File I/O
- Importance of releasing external resources

FINDING ADDITIONAL INFO ABOUT A FILE PATH

The **File** class has several methods which can be used to find extra information about the specified path. You have already seen **exists()** but here are a few of the available methods of this sort:

- **getName()** - returns name of file (just the name, not any path info)
- **getAbsolutePath()** - we saw this yesterday... returns absolute path of the File
- **exists()** - indicates whether the file or directory pointed to by File exists
- **isDirectory()** - indicates whether the path points to a directory
- **isFile()** - indicates whether the path points to a directory
- **length()** - size of file in bytes

USING THE FILE CLASS TO CREATE A DIRECTORY

One of the things the File class can be used for is creating a new directory.


```
File newDirectory = new File("myDirectory");

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

USING THE FILE CLASS TO CREATE A DIRECTORY

One of the things the File class can be used for is creating a new directory.

Create new `File` object with path of directory to create as the param.




```
File newDirectory = new File("myDirectory");

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

USING THE FILE CLASS TO CREATE A DIRECTORY

One of the things the File class can be used for is creating a new directory.

Create new **File** object with path of directory to create as the param.




Check if the directory already exists using the **File** object's **exists()** method before creating it.

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

USING THE FILE CLASS TO CREATE A DIRECTORY

One of the things the File class can be used for is creating a new directory.

Create new **File** object with path of directory to create as the param.



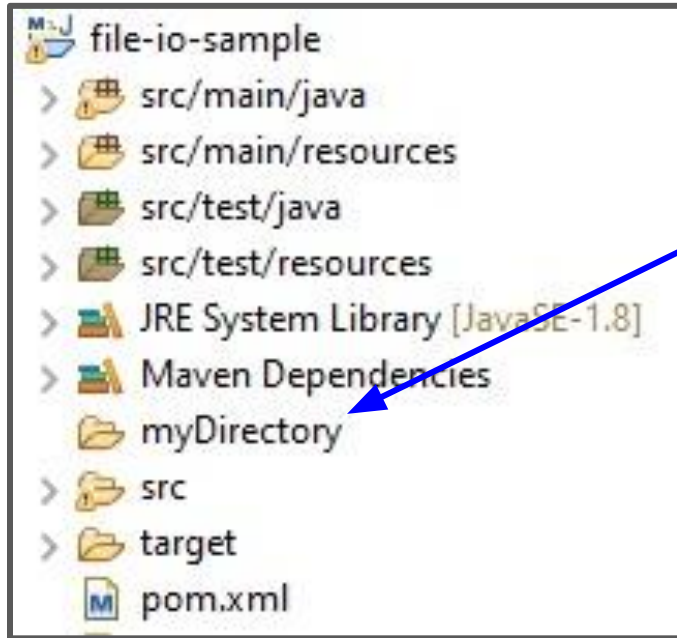
Check if the directory already exists using the **File** object's **exists()** method before creating it.

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

Call the **File** object's **mkdir** (remember that?) command to create the directory.

USING THE FILE CLASS TO CREATE A DIRECTORY

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



Notice that the directory has been created at the root level of the project.

USING THE FILE CLASS TO CREATE A FILE

We can also use the File class to create a file (also relative to the root of the project).

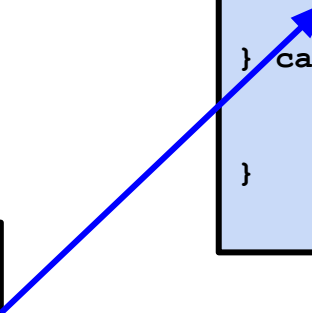
```
try {  
    File newFile = new File("myDataFile.txt");  
    newFile.createNewFile();  
} catch(IOException e) {  
    System.out.println("Exception occurred: "  
        + e.getMessage());  
}
```

USING THE FILE CLASS TO CREATE A FILE

We can also use the File class to create a file (also relative to the root of the project).

```
try {  
    File newFile = new File("myDataFile.txt");  
    newFile.createNewFile();  
} catch(IOException e) {  
    System.out.println("Exception occurred: "  
        + e.getMessage());  
}
```

Create new **File** object with path of file to create as the param.



USING THE FILE CLASS TO CREATE A FILE

We can also use the File class to create a file (also relative to the root of the project).

```
try {  
    File newFile = new File("myDataFile.txt");  
    newFile.createNewFile();  
} catch(IOException e) {  
    System.out.println("Exception occurred: "  
        + e.getMessage());  
}
```

Create new **File** object with path of file to create as the param.

Use the **createNewFile()** method of the **File** class to create the file on the filesystem

USING THE FILE CLASS TO CREATE A FILE

We can also use the File class to create a file (also relative to the root of the project).

Must use a **try-catch** block here since the **createNewFile** method of the **File** class declares it may throw an **IOException**.

Create new **File** object with path of file to create as the param.

```
try {  
    File newFile = new File("myDataFile.txt");  
    newFile.createNewFile();  
} catch(IOException e) {  
    System.out.println("Exception occurred: "  
        + e.getMessage());  
}
```

Use the **createNewFile()** method of the **File** class to create the file on the filesystem

USING THE FILE CLASS TO CREATE A FILE

We can also use the File class to create a file (also relative to the root of the project).

Must use a try-catch block here since the `createNewFile` method of the `File` class declares it may throw an `IOException`.

Create new `File` object with path of file to create as the param.

```
try {  
    File newFile = new File("myDataFile.txt");  
    newFile.createNewFile();  
} catch (IOException e) {  
    System.out.println("Exception occurred: "  
        + e.getMessage());  
}
```

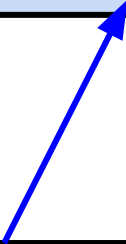
Catch and handle the possible `IOException`.

Use the `createNewFile()` method of the `File` class to create the file on the filesystem

USING THE FILE CLASS TO CREATE A FILE IN A DIRECTORY

The `File` class has an overloaded constructor which takes an extra parameter specifying the path in which the file or directory should be created.

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



Extra param which tells `File` to put the path in the second parameter into the directory at this path. Can be used to add a file to the directory or create a subdirectory.

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.

When more than one class is required to solve a problem, we typically refer to these classes as **collaborators**. In this case, the **File**, and **Printwriter** classes are collaborators.


WRITING TO A FILE

```
File newFile = new File("myDataFile.txt");
String message = "Appreciate, Elevate, Participate";

try {
    PrintWriter writer = new
        PrintWriter(newFile);
    writer.print(message);
    writer.flush();
    writer.close();
} catch (FileNotFoundException e) {
    System.out.println("File does not exist.");
}
```


WRITING TO A FILE

Text to write to file.



```
File newFile = new File("myDataFile.txt");  
String message = "Appreciate, Elevate, Participate";  
  
try {  
    PrintWriter writer = new  
        PrintWriter(newFile);  
    writer.print(message);  
    writer.flush();  
    writer.close();  
} catch (FileNotFoundException e) {  
    System.out.println("File does not exist.");  
}
```

WRITING TO A FILE

Text to write to file.

Create a
`PrintWriter` with
the `File` object.

```
File newFile = new File("myDataFile.txt");
String message = "Appreciate, Elevate, Participate";

try {
    PrintWriter writer = new
        PrintWriter(newFile);
    writer.print(message);
    writer.flush();
    writer.close();
} catch (FileNotFoundException e) {
    System.out.println("File does not exist.");
}
```

WRITING TO A FILE

Text to write to file.

Create a
PrintWriter with
the File object.

"prints" the text in
message to the
PrintWriter.

```
File newFile = new File("myDataFile.txt");
String message = "Appreciate, Elevate, Participate";

try {
    PrintWriter writer = new
        PrintWriter(newFile);
    writer.print(message);
    writer.flush();
    writer.close();
} catch (FileNotFoundException e) {
    System.out.println("File does not exist.");
}
```

WRITING TO A FILE

Text to write to file.

```
File newFile = new File("myDataFile.txt");  
String message = "Appreciate, Elevate, Participate";
```

Create a
PrintWriter with
the File object.

```
try {  
    PrintWriter writer = new  
        PrintWriter(newFile);  
    writer.print(message);  
    writer.flush();  
    writer.close();  
} catch (FileNotFoundException e) {  
    System.out.println("File does not exist.");  
}
```

"prints" the text in
message to the
PrintWriter.

What's this? Stay
tuned....

WRITING TO A FILE

Text to write to file.

```
File newFile = new File("myDataFile.txt");  
String message = "Appreciate, Elevate, Participate";
```

Create a
PrintWriter with
the File object.

```
try {  
    PrintWriter writer = new  
        PrintWriter(newFile);
```

```
    writer.print(message);
```

```
    writer.flush();
```

```
    writer.close();
```

"prints" the text in
message to the
PrintWriter.

```
} catch (FileNotFoundException e) {  
    System.out.println("File does not exist.");  
}
```

What's this? Stay
tuned....

Close the Printwriter.

WRITING TO A FILE

Text to write to file.

Create a `PrintWriter` with the `File` object.

"prints" the text in `message` to the `PrintWriter`.

What's this? Stay tuned....

Close the `PrintWriter`.

Catch the possible `FileNotFoundException`

```
File newFile = new File("myDataFile.txt");
String message = "Appreciate, Elevate, Participate";

try {
    PrintWriter writer = new
        PrintWriter(newFile);
    writer.print(message);
    writer.flush();
    writer.close();
} catch (FileNotFoundException e) {
    System.out.println("File does not exist.");
}
```

The diagram illustrates the process of writing to a file in Java. A central blue box contains a code snippet. Six yellow text boxes with black borders are positioned around the code, with blue arrows pointing from specific lines of code to the boxes. The boxes provide explanations for various parts of the code: the file name, the creation of the `PrintWriter`, the writing of the message, the flushing of the buffer, the closing of the `PrintWriter`, and the handling of a `FileNotFoundException`.

INTRODUCING... BUFFERS

A **buffer** is like a bucket to which text is initially written. It is only after we invoke the `.flush()` method that the bucket's contents are transferred to the file. **Printwriter** creates a buffered stream that gets flushed when the buffer is full or `.flush()` is manually called and the **Printwriter** is closed.

```
try (PrintWriter writer = new PrintWriter(newFile.getAbsolutePath())) {  
    writer.print(message);  
} catch (FileNotFoundException e) {  
    System.out.println("File does not exist.");  
}
```

Remember how we mentioned the try-with-resources block was created to avoid writing lots of repetitive, cluttered code? The try block in the previous example can be rewritten like this - the try-with-resources takes care of flushing the buffer and closing the **Printwriter** resource when the try block exits!!!!

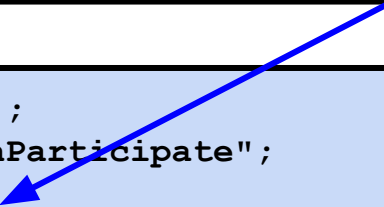
APPENDING TO A FILE

The previous example regenerates the file's contents from scratch every time it is 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.
- `PrintWriter(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).

FILE APPEND EXAMPLE

We set a `boolean` indicating whether to append based on whether the file being written to already exists.



```
File newFile = new File("myDataFile.txt");
String message = "\nAppreciate\nElevate\nParticipate";

boolean append = newFile.exists() ? true : false;
try (PrintWriter writer =
    new PrintWriter(new FileOutputStream(newFile, append))) {
    writer.append(message);
} catch (IOException e) {
    System.out.println("Exception: " + e.getMessage());
}
```

FILE APPEND EXAMPLE

We set a `boolean` indicating whether to append based on whether the file being written to already exists.

```
File newFile = new File("myDataFile.txt");
String message = "\nAppreciate\nElevate\nParticipate";

boolean append = newFile.exists() ? true : false;
try (PrintWriter writer =
    new PrintWriter(new FileOutputStream(newFile, append))) {
    writer.append(message);
} catch (IOException e) {
    System.out.println("Exception: " + e.getMessage());
}
```

We create the `PrintWriter` using a `FileOutputStream`, which is created using the `File` object and the `boolean` we created as the param which indicates whether or not to append.

FILE APPEND EXAMPLE

We set a `boolean` indicating whether to append based on whether the file being written to already exists.

```
File newFile = new File("myDataFile.txt");
String message = "\nAppreciate\nElevate\nParticipate";

boolean append = newFile.exists() ? true : false;
try (PrintWriter writer =
    new PrintWriter(new FileOutputStream(newFile, append))) {
    writer.append(message);
} catch (IOException e) {
    System.out.println("Exception: " + e.getMessage());
}
```

We use the `append` method of `PrintWriter` which will append if the stream it is created with is set to append.

We create the `PrintWriter` using a `FileOutputStream`, which is created using the the `File` object and the `boolean` we created as the param which indicates whether or not to append.