COMP1006/1406 - Tutorial 9

File I/O.

File I/O

There are several ways in which Java allows you read and write to files. There are classes for reading/writing binary files and there are classes for reading/writing text files. We'll focus on text files.

Whether you are reading from or writing to a file (or both), the following pattern should always be followed:

- open the file
- access the file (read from or write to the file)
- close the file

It is important that you always close a file when you are done reading it or writing to it. Can you think of any reasons why this is important?

If you want some extra resources for file i/o in Java, see Chapter 11 of Dr. Lanthier's notes. It contains notes and examples for writing/reading binary data, text data, objects and working with `File` objects.

1: Reading Text Files

The `FileReader` class is used to read data from a file character by character. Often, we use another class (like `Scanner` or `BufferedReader`) to wrap a FileReader object to make reading easier for us.

To create a `FileReader` object we specify the name of the file to open for reading in the constructor.

```
String fileName = "example.txt";
try{
   FileReader file = new FileReader( fileName );
}catch(java.io.FileNotFoundException e){
   System.err.println("File " + fileName + " was not found");
}
```

Notice that we had to wrap the creation of a FileReader object in a `try/catch`. This is because there might be a checked exception thrown in the constructor.

https://docs.oracle.com/javase/8/docs/api/java/io/FileReader.html#FileReader-java.io.File-

A BufferedReader object will read a text file character by character (returning the unicode value of the character) or will read a file line by line. It reads lines of the file as Strings and we'll use this class to read the contents of a file line by line. When you try to read a line (readLine()) beyond the end of a file, the method will return null. You can just read a file line by line until you get a null back from the method.

The BufferedReader wraps the FileReader class. We pass a Filereader as input to the constructor.

```
String fileName = "example.txt";
String line = null;

try{
  FileReader file = new FileReader( fileName );

  BufferedReader reader = new BufferedReader(file);

  line = reader.readLine();
  while((line != null) {
    System.out.println(line);
      line = reader.readLine();
  }

}catch(java.io.FileNotFoundException e){
  // The FileReader constructor might throw this exception
  System.err.println("File " + fileName + " was not found");

}catch(java.io.IOException e){
  // Reading from the BufferedReader might throw this exception
  System.err.println("IOException thrown : " + e);
}
```

Notice that we now that we now also try to catch a java.io.IOException. This exception might be thrown from the BUfferedReader when reading data.

http://docs.oracle.com/javase/8/docs/api/java/io/BufferedReader.html

➨ Download the Files.java program; compile and run the program. Notice that is simply reads itself and prints the contents of the file to the screen. Modify the file so that all comments and blank lines are stripped out of the output to standard output.

2: Writing Text Data

The Write class in Java has several subclasses to let you write data to text files. The BufferedWriter class is used for writing lines of text, the FileWriter is used for writing characters, and the PrintWriter class lets us write strings (just like the PrintStream class lets us print to the screen).

As with reading, we use a basic writing class to write single characters to a file, the FileWriter class, and wrap it another class that makes writing more than just a character at a time simpler.

```
String fileName = "example.txt";
try{
  FileWriter file = new FileWriter( fileName );
  ...
}catch(FileNotFoundException e){
  System.err.println("Error: could not find file \"" + fileName + "\"");
}catch (IOException e) {
  System.out.println("Error: Cannot read from file");
}
```

We will consider the `PrintWriter` class for outputting text to a text file. Look at the API for the methods that it provides to us.

`http://docs.oracle.com/javase/8/docs/api/java/io/PrintWriter.html`

You should notice that it has the same methods that we used for printing to standard output: `print` and `println`. Further, we can pass any object to these methods and it will print the results of calling `toString()` on the objects to the file.

➨ Modify your `Files.java` program to do the following:

- Use a command line argument to specify the filename of the file to read. Suppose it is called `file.java`.
- Write a new file called `file-stripped.java` that is a copy of the `file.java` file but without any comments and blank lines.

Extra Reading

`http://people.scs.carleton.ca/~lanthier/teaching/COMP1406/Notes/COMP1406_Ch11_FileIO.pdf`