Hyperiondev

TASK

Java - IO

Visit our website

Introduction

WELCOME TO THE JAVA - INPUT AND OUTPUT TASK!

The Java programs we have written so far store data in variables. As soon as we end the program, the data in the variables are lost. In this task, you are going to be introduced to file input and output: a way in which we can write program data to an external storage medium (i.e. your hard drive or USB flash drive).

READING A FILE

Files are useful for storing and retrieving data. There are several ways to read from a file. One of the simplest ways is to use the `Scanner` class from the `java.util` package. The constructor of the `Scanner` class can take a `File` object as input. To read the contents of a text file at the path "C:\\test.txt", we would need to create a `File` object with the corresponding path and pass it to the `Scanner` object:

```
import java.util.Scanner;
import java.io.File;
import java.io.FileNotFoundException;


try {
   File x = new File("C:\\test.txt");
   Scanner sc = new Scanner(x);
}

catch (FileNotFoundException e) {

}
```

***Note**: We surrounded the code with a **try-catch** block because there's a chance that the file may not exist. Try/catch blocks are used for error handling. If the code in the try block throws an exception (i.e. an error occurs), the code in the catch block is executed. You will learn more about try-catch blocks in a later task but for now, you can just follow the format above. For more about Java try-catch blocks, you can have a look **here**.*

The `Scanner` class inherits from the ***Iterator interface***. The Iterator interface is a special type of class that other classes can inherit from that allows one to iterate through a collection of items (don't worry too much about this yet either. You'll

learn about this in greater detail in a later task). To iterate means to go through the items one at a time. The `next()` method is used to do this. Since the `Scanner` class inherits from the Iterator interface, you can iterate through a file using the `Scanner` class. We can use the `Scanner` object's `next()` method to read the file's contents:

```
try {
   File x = new File("C:\\test.txt");
   Scanner sc = new Scanner(x);
   while (sc.hasNext()) {
      System.out.println(sc.next());
   }
   sc.close();
}

catch (FileNotFoundException e) {
   System.out.println("Error");
}
```

The file's content is output word by word because the `next()` method returns each word separately. If you wanted to output line by line, you can use the methods `hasNextLine()` and `nextLine()` instead.

Note: *It is always good practice to close a file when finished working with it. One way to do this is to use the Scanner's* ***`close()`*** *method.*

CREATING FILES

`Formatter`, another useful class in the `java.util` package, is used to create content and write it to files. Example:

```
import java.util.Formatter;

public class MyClass {
   public static void main(String[] args) {
      try {
         Formatter f = new Formatter("C:\\test.txt");
      }
      catch (Exception e) {
         System.out.println("Error");
      }
   }
}
```

This creates an empty file at the specified path. If the file already exists, this will overwrite it. Again, you need to surround the code with a *try-catch* block, as the operation can fail.

WRITING TO FILES

Once the file is created, you can write content to it using the same Formatter object's `format()` method. Example:

```
import java.util.Formatter;

public class MyClass {
   public static void main(String[] args) {
      try {
         Formatter f = new Formatter("C:\\test.txt");
         f.format("%s %s %s", "1", "John", "Smith \r\n");
         f.format("%s %s %s", "2", "Amy", "Brown");
         f.close();
      }
      catch (Exception e) {
         System.out.println("Error");
      }
   }
}
```

The `format()` method formats its parameters according to its first parameter. `%s` means a String and gets replaced by the first parameter after the format. The second `%s` gets replaced by the next one, and so on. So, the format `%s %s %s` denotes three Strings that are separated with spaces. The code above creates a file with the following content:

```
1 John Smith
2 Amy Brown
```

Notes: `\r\n` *is the newline symbol in Windows. Don't forget to close the file once you're finished writing to it!*

Compulsory Task 1

Follow these steps:

- Create a new java file called **Poetry.java**
- This task is going to be based on your Level 1 Capstone Project II where you created the Caesar cipher.
- Recreate your project from JavaScript into Java.
- Include file input-output functionality in your code so that the program can read a text file, encode it, and output the encoded message into a new text file
- Use your program to read the content of the text file **poem.txt**. For each line in **poem.txt**, write a new line in a new text file **encodedPoem.txt** that is encoded by the cipher.
- Compile, save and run your file.

Compulsory Task 2

Follow these steps:

- Modify your **Poetry.java** file to do the following:
- If the letter is coded to a vowel, make it a capital letter.
- Write your new encoded poem to a text file called **capitalVowels.txt**.
- Compile, save and run your file.

Compulsory Task 3

Follow these steps:

- Modify your **Poetry.java** file to do the following:
- Take **capitalVowels.txt** and reverse the characters of each line of the file
- Then, decode the text back to its original letters
- Finally, write this text to a new file called **reversePoem.txt**.
- Make sure that the capitalised vowels go back to their original letters!
- Compile, save and run your file.

Rate us

Share your thoughts

HyperionDev strives to provide internationally-excellent course content that helps you achieve your learning outcomes.

Think that the content of this task, or this course as a whole, can be improved or think we've done a good job?

Click here to share your thoughts anonymously.