Hyperion Development

Task: Files

www.hyperiondev.com

Introduction

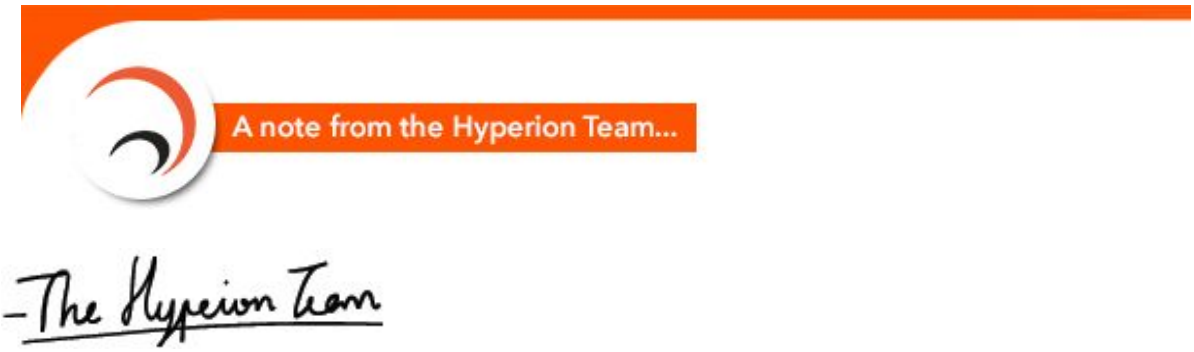
Overview

In the previous tasks, we wrote code which could take input from a user using the function ''JOptionPane.showInputDialog()''.

We also could display contents onto the console using the built-in function ''System.out.println()''. What happens if we want to write data to a storage medium that stores the data permanently?

The Java programs we have written so far store data in a variable and as soon as we end the program, the data is 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, and there are a number of 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:

```
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.*

The Scanner class inherits from the Iterator, so it behaves like one. 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 contents are output word by word, because the **next**() method returns each word separately.

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.

Note: *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 mean 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.

Note: *\r\n is the newline symbol in Windows.*

The code above creates a file with the following content:

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

Note: *Don't forget to close the file once you're finished writing to it!*

Instructions

First read example.java, open it using jGRASP (Right click the file and select 'Open with jGRASP").

- In this folder there is a file called example.java
- Open this folder using the JGRASP program. You can either do this by right clicking on the example.java file, going to the 'Open with' menu, and selecting JGrasp.exe. Alternatively, you can run the JGRASP program on your computer, go to the top left corner of the program, select File->Open and navigate to example.java on your hard drive and double click on it to open it.
- Once example.java is open in JGRASP please read all of its content very carefully. This will help you understand the basic structure of a Java program.
- There is a compulsory task at the end of the example.java document. The instructions of what to do to complete the task can be found here. You must complete this exercise to continue onto the next task of this course.

Compulsory Task 1

Follow these steps:

- Create a new java file called MyFile.java
- Write code to read the content of the text file **input.txt**. For each line in **input.txt**, write a new line in the new text file **output.txt** that computes the answer to some operation on a list of numbers.
- If the input.txt has the following:
  Min: 1,2,3,5,6
  Max: 1,2,3,5,6
  Avg: 1,2,3,5,6

  Your program should generate output.txt as follows:
  The min of [1, 2, 3, 5, 6] is 1.
  The max of [1, 2, 3, 5, 6] is 6.
  The avg of [1, 2, 3, 5, 6] is 3.4.
- Assume that the only operations given in the input file are min, max and avg, and that the operation is always followed by a list of comma separated integers. You should define the functions min, max and avg that take in a list of integers and return the min, max or avg of the list.
- Your program should handle any combination of operations and any length of input numbers. You can assume that the list of input numbers are always valid integers and that the list is never empty.
- Compile, save and run your file.

Compulsory Task 2

Follow these steps:

- Modify your MyFile.java file to do the following:

- Change your program to additionally handle the operation "px" where x is a number from 10 to 90 and defines the x percentile of the list of numbers. E.g.:

  Input.txt:
  Min: 1,2,3,5,6
  Max: 1,2,3,5,6
  Avg: 1,2,3,5,6
  P90: 1,2,3,4,5,6,7,8,9,10
  Sum: 1,2,3,5,6
  P70: 1,2,3

- Your output.txt should read:
  The min of [1,2,3,5,6] is 1.
  The max of [1,2,3,5,6] is 6.
  The avg of [1,2,3,5,6] is 3.4.
  The 90th percentile of [1,2,3,4,5,6,7,8,9,10] is 9.
  The sum of [1,2,3,5,6] is 17.
  The 70th percentile of [1,2,3] is 2.

- Compile, save and run your file.

Things to look out for

1. Make sure that you have installed and setup all programs correctly. You have setup **Dropbox** correctly if you are reading this, but **jGRASP or Java** may not be installed correctly.
2. If you are not using Windows, please ask your tutor for alternative instructions.

Still need help?

Just write your queries in your comments.txt file and your tutor will respond.

Task Statistics

Last update to task: 19/02/2016.
Authors: Jared Ping.
Main trainer: Umar Randeree.
Task Feedback link: Hyperion Development Feedback.