TASK

External Data Sources — Files

Visit our website

Introduction

WELCOME TO THE 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).

Get in touch

Connect for support

Remember that with our courses, you're not alone! You can contact your mentor to get support on any aspect of your course.

The best way to get help is to login to **www.hyperiondev.com/portal** to start a chat with your mentor. You can also schedule a call or get support via email.

Your mentor is happy to offer you support that is tailored to your individual career or education needs. Do not hesitate to ask a question or for additional support!

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:

```
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. Read more about Java try-catch blocks **here**. You will cover try-catch blocks in more detail in a later task.*

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. 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 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. 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
```

Absolute vs Relative File Paths

File paths may either be relative or absolute. Absolute file paths are expressed with reference to the root directory, whereas relative file paths are expressed with reference to the process' current working directory. You can refer to **Wikipedia's table**, covering the symbols for these directories on various operating systems. When shipping programs with files they will read from or write to, rather than using absolute file paths, you can use relative file paths to ensure your program finds the files independent of your development environment.

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 **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 [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 a list of comma-separated integers always follows the operation. 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 [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.

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.