CIS 255 • Java Programming
Laboratory Assignment 5: File Input and Output

In this laboratory assignment you will work with Java file input and output syntax to generate lists of hot days and cold nights in a given month.

Background

A local weather office stores information about the high and low temperatures each day for a month in the file `MonthTemps.txt`, where each line has information about one day's temperatures:

day high low

An example line would be this (on the 28th, the high was 59, and the low was 28):

```
28 59 28
```

The weather office wants the data for each day in which the high temperature was at least 80 degrees written to the file `HotDays.txt`; it also wants the data for each day in which the low temperature was below 30 degrees written to the file `ColdNights.txt`. The month represented in the text file could be any month, so you won't know ahead of time whether the number of days is 28, 29, 30, 31, or a partial month with only a few days.

Step 1: Download the Starter File and Sample Input File and Launch TextPad

On Blackboard, click the link for the starter file to download it. Make sure you insert your first initial and last name and the laboratory assignment number where indicated (**FLast_Lab05.java**) and that you specify the file location. You'll also need to download the sample `MonthTemps.txt` file to the same location where you save the starter file. Once you have downloaded the file, open it by launching TextPad and using the Open command or by right-clicking the file in Windows and selecting Open With > TextPad.

If you place an input file in a location other than where your source code is, you'll need to include the path to that location in the quotation marks with the file name when declaring the `File` object.

Step 2: Customize Your Code and Add Import Statements

Edit the comments at the top of the code to include your name, the course number, the laboratory assignment number, and a brief description of the program's contents. You may also wish to include the date (perhaps after the laboratory assignment number). Also, modify the class name to match your file name. As you continue your code, you may go ahead and add comments to the statements or groups of statements that you believe deserve additional explanation, or you may wait until after you have written your entire program to add comments.

In dealing with file input and output, you'll need to add the wildcard `import` statement for all of the classes in the `java.io` package and the `import` statement for the `Scanner` class used for the input file. The program should display an error message if the input file does not exist; if you want this error message to appear in a dialog box, you'll need the `import` statement for `JOptionPane` as well.

Also, modify the header of the `main` method to indicate that exceptions of type `IOException` should be thrown so that you will not have to write more complicate exception-handling syntax.

A program that involves file input and output will not compile without some indication as to how the exceptions are to be handled. This is because the exceptions in the `IOException` family are *checked exceptions*; the other exceptions that we've seen are known as *unchecked exceptions*.

Step 3: Declare Variables for Primitive Values

Declare one variable each for the day, high temperature, and low temperature (all whole numbers) to be read from each line in the file. Since you'll be writing the values to the appropriate output file when necessary, you won't need to maintain these values throughout the program, so you can reuse them to go from one record to the next; in other words, you don't have to declare 31 day variables, 31 high temperature variable, and 31 low temperature variables.

You may also want to declare constants for the criteria for a hot day and a cold night, respectively.

Step 4: Declare Objects to Open the Files

For the input file, declare an object of type `File` that uses the file name `MonthTemps.txt` (in quotation marks) as the argument to the new `File` object. Make sure that the program exits with an error message if it is unable to open the file. After you confirm that the file does exist, pass the `File` object to a new `Scanner` object. (See the section "Checking for a File's Existence" in your textbook.)

For the output files, declare two objects of type `PrintWriter`: one using the file name `HotDays.txt` as the argument to its object, and the other using the file name `ColdNights.txt` instead. Assume that it's okay to overwrite the contents of each file if it already exists (don't append to the file). See the section "Using the `PrintWriter` Class to Write Data to a File" in your textbook for more details.

You must have a separate `PrintWriter` object for each output file; you cannot use the same object for both files. The object variable names must be different.

If you want to add to contents of an existing file instead of overwriting it, create a `FileWriter` object for that file first; then, place the `FileWriter` variable name in the parentheses of the statement that declares the `PrintWriter` object.

Step 5: Use a Loop to Read the Input File's Contents

Since you don't know how many days will be represented in the input file, you'll need a loop that reads as long as there are more unread contents to be processed. This can be accomplished by invoking the method `hasNext()` on your `Scanner` object as the Boolean expression for your loop. (See the section "Detecting the End of a File" in your textbook.)

Inside the loop body, use three calls to `nextInt()` to read the next three values from the file into the variables for the day, high temperature, and low temperature.

Then, use an `if` statement to determine if the data represents a hot day; if so, call the `println` method to send all three values for that day to the first output file. Write another `if` statement to determine if the data represents a cold night, with another `println` method call that sends all three values to the second output file. If the data represents neither a hot day nor a cold night, it should not be written to either output file.

Although highly unlikely, a day might have both a high temperature of at least 80 and a low temperature below 30. Use separate `if` statements instead of `else if` to make this work.

You must include spaces (in quotation marks) between the values to be written on a line of output to a file; otherwise, the values will run together.

Step 6: Close the Files

Invoke the `close()` method on all three file-related objects (the `Scanner` object and the two `PrintWriter` objects) to ensure that the program terminates properly.

If you do not close an output file, the text to be sent to the file may remain in the output buffer without being written to the file before the program ends.

Step 7: Compile Your Code and Execute the Byte Code

To determine if there are any syntax errors in your source code, and to generate the byte code once your source code is free of syntax errors, select the item **Compile Java** from the **External Tools** submenu of the **Tools** menu. The results of the build process will appear in the **Tool Output** area below your code. If the Tool Output area indicates that there are syntax errors in your code, you may double-click the line that describes each syntax error to see the line that contains the error (or, in some cases, the line after the error).

Once you receive a message in the Tool Output area that the tool completed successfully, you're ready to test your program. In the **External Tools** submenu of the **Tools** menu, select **Run Java Application**. If the program is able to read from the sample input file and generate the two output files, you will find the new files in the same location as your code file; no output will appear in the console other than the text "`Press any key to continue...`". Open these files, verifying that the hot days and cold nights were written properly to their respective files. You may want to modify the values in the sample input file to make sure that it handles varying sets of data properly (for example, longer or shorter months, a larger or smaller number of hot days or cold nights, etc.).

Step 8: Add Inline Comments and Submit Your Code

Make sure that your code has **three or more inline comments** near the statements that you believe warrant the additional description (especially those related to files).

Follow the instructions on the course website to attach and submit your source code.

Make sure you complete the submission process! If you attach your file without submitting the assignment, your file will not be received!