CS112 - Spring 2024

Lab20

Instructor: Paul Haskell

INTRODUCTION

Back to the big labs! In this lab, you will work with some of the Random library components we learned about. You will all do some experiments with Bubble Sort, and you will write an automated test program. You will do some File I/O programs also.

Bubble Sort

The **Lab20** directory in the CourseInfo repository has a Java Bubble Sort program for you, called **Bubble.java**. This program generates 10,000 random numbers, sorts them with a bubble sort, saves them to a file called "**sort.txt**", and prints to `System.err` the number of seconds required to perform the sort.

Your next task is to translate the Java Bubble Sort to Python, and write a program called **Bubble.py** . Use this program to generate 10,000 random numbers, sort them, and print out how much time was required for the sort. (Do not use Python's built-in sort() methods--they use a smarter sorting algorithm than Bubble Sort.) You do not have to save the sorted values to a file.

Bubble.py should just print the number of seconds as a floating point number, with no units of time or text description, e.g.

20.593817

Create (probably by hand) a text file called **speed.txt** and in it write[1]

Java <<# of seconds required on your computer for Bubble.java to sort 10k elements>>

Python <<# of seconds required on your computer for Bubble.py to sort 10k elements>>

I will collect and graph this data for all of us to review.

[1] Remember – don't put the "<<" and ">>" in your **speed.txt** file. These symbols just mean "substitute your actual answer in here".

Test Software

Next, please write an automated tester for **Bubble.java**, called **SortTest.java** . This program should accept the name of an input file in `args[0]`. The program reads the file, ensure it contains the expected number of values, and ensure the values in the file are nondecreasing. If the file is good, print to `System.out`

PASS

If the file does not have the right number of elements, print

FAIL incorrect element count

If the file does have the right number of elements, but the file's elements are not sorted properly, print

FAIL incorrect sort

Testing the Test Software

Your test software is software, and should be tested itself! Please create two "good" test files (correct number of elements and properly sorted) and name them "**good1.txt**" and "**good2.txt**". Please create two "bad" test files: the first with an incorrect number of elements and the second with the correct number of elements but with improperly sorted data. Name those files "**bad1.txt**" and "**bad2.txt**".

You don't have to write programs to generate these test files—you can use a text editor. Please use the files to test your **SortTest.java** program!

List Merge

The next problem is **Merge.java**. Three command line arguments give the names of **three** input text files, each of which contains an already-sorted (increasing order) list of zero or more integers. The program should read the three lists (into ArrayLists?) and print out to `System.out` a merged sorted (increasing order) list that combines all elements of the three input lists.

Since the input lists are sorted already, you must combine them in sorted order without doing another sort! Hint: just look at the first not-yet-output elements of each list. But be warned: not all the lists are necessarily the same length. Please do not do a sort operation to merge the lists—we will need your **Merge.java** when we develop sorts ourselves.

Error handling is important:

- if there is a problem with the user input, print 'ERROR' to `System.err`, along with an explanation of the error
- Errors could include too few input arguments, or a filename that is not found. An empty file is not an error.
- if there is an error, print nothing to `System.out` … only to `System.err`

CSVWriter

For the biggest program this week, you will create a tool that I had to create to help grade the first big class project!

"CSV" stands for "comma separated values", which is the name of a format for text files that contain multiple lines, each line with the same number of entries, with each line's entries separated by commas, e.g.

```
first,second,third
ready,set,go
Giants,Niners,Warriors
mocha,latte,cappuccino
```

CSV files are used because they are "plain text", i.e. readable by humans if you just print out all the characters. But this format is also readable directly by spreadsheet programs such as Microsoft Excel. This format is an "interchange format" that allows data to be transferred between different spreadsheet programs or between spreadsheet and non-spreadsheet programs.

You will write a program called **CSVTester.java**. In this file, write a class called `CSVWriter`. Please derive `CSVWriter` from `FileWriter`. **CSVTester.java** also should contain a `class CSVTester`, with just a `main()` method, which tests `CSVWriter`.

Ok, what should class `CSVWriter` do? It should have a constructor that takes a `File` input and another that takes a `String` input—the `String` holds the pathname of the `File` to be written.

CSVWriter should implement a `void writeln(String[] stringsForALine)` method. This method uses the underlying `FileWriter write()` method to write each of the `stringsForALine` to the `File` specified in the constructor. And of course, the `CSVWriter writeln()` function should put a comma after every `String` in every line, except not after the last `String` in each line. The last `String` in each line is simply followed by a NEWLINE.

That's it? Not quite, but that's most of it. The user of `CSVWriter` will want to write multiple lines to a file. The spreadsheet programs that read the CSV files insist that every line have the same number of entries. Please create a new `Exception` class (you pick the name), derived from `class Exception`, and have your `writeln()` method throw your new exception if the user calls `writeln()` with some number of strings per line that differs from the number in all previous `writeln()` calls. The text in your new `Exception` type should give the expected number of strings per line (from earlier calls to `writeln`) and also the number in the troublesome `writeln` call.

Your **CSVTester.java** program should keep reading the **keyboard** input until the keyboard input terminates (remember how to check for that?). The program should take each line of keyboard input, parse out the separate words (separated by one or more spaces), and then call `CSVWriter.writeln()` with a list of all the words for each line. Please have the program output to a file called "**CSVTester.csv**" . And as hinted above, if the user does not always enter the same number

of words per line, **CSVTester.java** should fail with your new `Exception`. (The autograder can tell that the program failed and can see the text in your exception—of course your testing should try this also.)

Does your program need any special handling for inputs in double quotes (e.g. treating "cat dog" as one word rather than two)? No! The special double-quote handling that we have talked about is a feature of your Operating System (MacOS or Windows or Linux). It is not done by Java and it is not needed from your program.

Still having fun? One last exercise. Create yet another new `Exception` type, derived from the new `Exception` type you defined previously. This can be called the `EmptyLineException`. Throw this exception if the user enters an empty line of text. Now have `class CSVTester` catch this new `Exception` type, and use it to close the `CSVWriter` and terminate the program cleanly, with proper data output and with no error messages.

To review:

- If user keeps entering same number of words per line and then terminates input, `CSVWriter` writes each line's words with comma separators
- If user changes the number of words per line, `CSVWriter` throws your new `Exception`, `CSVTester` does not catch it, and the program crashes with an error message
- But if user enters a line with zero words, even if there is more input afterwards, then `CSVWriter` throws an `EmptyLineException`. `CSVTester` catches it, closes the `CSVWriter`, and exits the program

Phew!

Reminder

Put all your files in **Lab20** and push to GitHub before the deadlines:

- Bubble.py, speed.txt, SortTest.java, and the four test files are all due at the end of class. Get help if you need it!
- Merge.java and CSVWriter.java must be turned in before 11:59pm Monday April 15th.

Conclusion

These programs give good practice thinking about and writing test software, and they are good preparation for our upcoming work on more advanced sorting techniques. Your **SortTest.java** will be useful to you then.

Grading Rubric

Bubble.py is worth 10 points if it runs properly, prints out a time to sort, and if the time is reasonable.

speed.txt is worth 5 points if it is present and contains the requested data.

SortTest.java is worth 15 points: 3 points for each of five test cases

good1.txt, **good2.txt**, **bad1.txt**, and **bad2.txt** together are worth 10 points if they all exist and are created properly.

Merge.java is worth 15 points: 3 points each for 5 test cases. No points if your **Merge.java** does sorting

CSVTester.java is worth 30 points: 4 points for each of 5 test cases, and 0-10 points based on the graders' evaluation of software quality and design.