CS206 Introduction to Data Structures

Lab 2

UNIX, I/O

Friday Sep 18

Work on the following for about 90 minutes.

What to Hand In:

Answers to questions 5, 10 and 11 in the UNIX section

The last piece of code you got to in writing in either Exercise 1 or 2.

You can do this by taking photos, or copy/paste or however it is easiest for you. Send email with these two things to gtowell206@cs.brynmawr.edu

UNIX — java at the command line; making and removing directories and files

Do the following:

1. Open a UNIX terminal window Applications / System Tools / MATE term
2. Determine your current directory (use the pwd command)
    1. Note that the UNIX prompt mentions this directory name
3. Change your directory to the one containing you VS code working directory for Lab1. The command to do so is likely be `cd cs206`.
4. Change the directory to the one for Lab 1. The command should be `cd Lab1`
5. Confirm that you are in the directory containing the Lab1 code. (How?)
6. Running a Java program has two steps: compiling and running
    1. Compile a Java program
       `javac HelloWorld.java` (This will compile HelloWorld.java and all java files used by HelloWorld.java)
    2. Run your program
       `java HelloWorld`
7. The javac command created a new file — HelloWorld.class which is then used to run your program. Before running your program you must (re)compile. If you do not, you may run an old version of your program. (Also before compiling you must save …) Therefore the process

to run a java program from the command line is:

   A. Save all java files
   B. javac XXX.java
   C. java XXX

8. Note that VS Code does these three steps for you when you hit "run".

9. Two new UNIX commands: `cat` and `rm`:

10. `cat` displays a file in the terminal window Try the following:

    1. cat HelloWorld.java

    2. cat HelloWorld.class

        1. You should not see anything reasonable for this. Why?

11. `rm` removes files. Be warned, rm is permanent and irreversable. Files deleted using rm are gone. Do the following:

    1. rm HelloWorld.class

        1. What did this command do? Why is it always OK to delete class files

Exercise 1: File Input/Output (I/O): The BufferedReader class and CSV files

Assignments in this course will typically deal with lots of data. This data will often be read by your program from a data file. Java has several ways to access files and read data from them. The Scanner class is one of the simplest; however it is buggy and slow. Therefore Scanner should not be used in this class other than for user input. Rather, use BufferedReader. You have already seen the use of BufferedReader for file reading (in the previous lab) and homework.

Many of the data files used in this class will be in CSV format where CSV is an acronym for Comma Separated Value. CSV files are a common format for exchanging data between spreadsheets. For example,

```
15,zip,city,state,population,males,females,
49079,Paw Paw,MI,13606,6764,6842,
49080,Plainwell,MI,15802,7838,7964,
```

The goal of this section of the lab is to write a program that reads a CSV text file and prints out is contents, item by item. Also, print the number of lines and the total number of items. For instance, for the 3 lines above you would print:

```
15
zip
city
…
7964
```

```
      Lines: 3
      Items: 19
```

The CSV file you should read is in /home/gtowell/Public/206/a2/testZip.csv

First, start a new program (java class) CSVReader.java by clicking on the new file icon next to Lab2 in the left side of your screen. Name the file CSVReader.java. Enter the following program (lots of typing here, or just copy/paste):

```
import java.io.BufferedReader;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
public class CSVReader {
    public CSVReader() {
    }

    public void readAndCount(String filename) {
        int lineCount=0;
        int fieldCount=0;
        try (BufferedReader br = new BufferedReader(new 
FileReader(filename));) {
            String line;
            while (null != (line = br.readLine())) {
                String[] splitLine = line.split(“,"); // break up a line by 
commas
            }
            System.out.println(lineCount + " " + fieldCount);
        } catch (FileNotFoundException e) {
            System.err.println("Could not open file " + e);
        }  catch (IOException e) {
            System.err.println("Problem reading " + e);
        }

    }

    public static void main(String[] args) {
        new CSVReader().readAndCount(“A FILE NAME”);
    }
}
```

If you are working on your own machine copy this file as follows:

1. Open a terminal (or power shell, etc)
2. Change your directory to the one which you created in the previous step
3. execute the following
   ```
   scp YOURNAME@powerpuff.cs.brynmawr.edu:/home/gtowell/
   Public/206/a2/testZip.csv .
   ```
   1. the final dot . is important.
   2. YOURNAME should be replaced by your CS department login name.

This program is incomplete with respect to the goals given above in that it does not do the requesting printing, nor does it do the requested counting. However, it does read the entire file and, more importantly, it breaks up each line in the file by commas. Extend the program to add the requisite counting and printing.

Exercise 2: ArrayLists and reading files: The balance sheet of a Day Trader

Day Traders buy and sell stocks, frequently. Some will make more than 1000 trades in a day. At the end of the day, they need to know their "position" — that is the number of shares they hold of which stocks. In this lab you will implement a system for tracking day trading positions, and printing out those positions at the end of the day. (Day traders also need to know their cash position, profit/loss etc. We are only going to deal with the number of shares they hold.)

Suppose that there are day traders who record all of their trades in a single file that has one transaction per line. Each line is of the form:

symbol amount

where:

symbol: is a stock symbol (for example IBM)
amount: is an integer, either positive or negative (for example -400).

For example, assuming that the trader starts the day with nothing, and that the file contains:

```
IBM,+200
MSFT,+100
IBM,-400
```

then at the end of the day their position would be:

```
MSFT 150
IBM —200
```

Yes, negative positions are allowed. In the stock market this is called "short selling". For details, watch the movie "The Big Short".

For this exercise write a Java program to read all of the transactions from the file

```
/home/gtowell/Public/206/lab02/bigtrades.txt
```

and then print out all of the non-zero positions. You should read the transaction file only once, so you will need to store information in a data structure. Specifically, use an ArrayList.

For easier debugging, there are much smaller versions of this file in:
`/home/gtowell/Public/206/lab02/trades.txt`

```
/home/gtowell/Public/206/lab02/microtrades.txt
```
microtrades.txt has the data in the example above

You can use the instructions for exercise 1 above to copy files, if you are working on your own machine.

To complete this exercise do the following (this is a suggested set of steps, you can follow your own path instead):

1. Create a class (give it a likely name like Position, you may use any name that works for you) that has instance variables for holding a stock symbol and the number of shares. It should have accessor methods as needed.
2. Create a second class with a likely name like DayTrader. Within that class create an ArrayList that holds instances of Position.
3. Assume that the day trader starts with nothing, so the arraylist is initially empty
4. Read the file line by line and split the line into pieces (using much the same code as in exercise 1 above).
5. With a line:
    1. Create an instance of Position containing the data on that line
    2. Search through the ArrayList of existing positions for one with the same symbol name.
    3. If one exists, update it with the number of shares in the just created Position instance. Otherwise add the Position to the ArrayList.
6. After reading all lines in the file, print all entries in the ArrayList that have a non-zero number of shares.