CS206 Introduction to Data Structures

Lab 2

UNIX, Runtime Errors, I/O

Thursday, Jan 30

When you are complete (or at least ready to leave), turn in this page with your name and the line and item counts from the CSVRreader program described below.

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

Runtime Errors

The best way to understand runtime errors is to write code that crashes!

Start Visual Studio Code

1. Select File Menu / Close Folder
2. Select File Menu / Open Folder
    1. Click on cs206 in the main folder list
    2. Click on the starred folder icon in the upper right (UNIX). On Macs this is "add folder" in lower left.
        1. Name the new folder "Lab2"
        2. Click on "Create"
    3. Click on "OK"
3. In the listing on the left side of the screen find Lab2 and click on the "new file" icon
4. Name the new file "Crash1.java"

Please note that these directions may be slightly different from those I have given previously based on my evolving understanding of Visual Studio Code.

Enter the program below

```
public class Crash1 {

	static int[] a = { 10, 20, 30, 40, 50 };

	public static void main(String[] args) {

		for (int i = 0; i < a.length; i++) {
			System.out.println(a[i]);
		}

		System.out.println("Done printing the array!");

	}
}
```

Run the program. (click on the bug on the left side of the VSC window, then tap on "run with java")

This is the correct version. You will get the output shown below.

```
10
20
30
40
50
Done printing the array!
```

Now, let's break it.

Change the termination test in the for-loop to the following:

i <= *a*.length

Can you tell what's going to happen? Why?

Run the program. You should now see the output shown below:

```
10
20
30
40
50
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 5
      at Crash1.main(Crash1.java:8)
```

You got a runtime error in your program. After printing out the last element in the array, it got the error. Why? Did you guess correctly?

Let's try and parse the error message. Runtime errors are called *exceptions* in Java and they throw Exceptions. If you read the first line, it is telling you that there was an exception in the thread "main". In computing, each set of instructions running on the computer is called a thread. At any point there could be several threads of computation running. So, this exception occurred in the "main" thread (that is your main() method).

Further, it is also telling you that the error, or exception, that occurred is a known type of error. It is called java.lang.ArrayIndexOutOfBoundsException. Just by looking at the name, you should be able to understand a lot about the problem. Specifically, the loop runs beyond the bounds of the array, to an index 5, which in an array of five elements does not exist. In fact, the index number that caused the exception is also shown (:5), as is the line at which the exception occurred:

Crash1.main(Crash1.java:8)

That is, at line 8 of Crash1.java (the actual line number in your code may vary) was where the program tried to access an array element that doesn't exist. This is line 8:

```
System.out.println(a[i]);
```

Since I=5 and a[5] does not exist, you get the ArrayIndexOutOfBoundsException.

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. 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/Public206/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, sorry):

```
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”);
    }
}
```

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.

Exceptions — Part 2: (Only if you have time)

Write a program to input a number and output its square root.

Create a new program within VSC (by clicking on the new file icon next to Lab2 in the left side of your screen) with the code shown below. Name the file Crash2.java.

```
public class Crash2 {

    public void doIt()
    {
        BufferedReader br = new BufferedReader(new
InputStreamReader(System.in));
        while (true) {
            System.out.println("Enter a number: ");
            try {
                int integ = Integer.parseInt(br.readLine());
                System.out.println("The number you entered times 4  is " +
(integ*4));
            } catch (IOException ioe) {
                System.err.println("Problem: " + ioe.toString());
            }
        }
    }

    public static void main(String[] args) {
        new Crash2().doIt();
    }
}
```

Hit the “run” button. “Enter a number: “ will appear in the bottom panel. The program then waits for your input. The code shown above repeatedly prompts the user for a number and then computes and outputs the square root of that number. The number is actually input as a string. You use the Java function:

Integer.parseInt(<string>)

to extract a number from the input string. There is also a Double.parseDouble() if you want to accept floating-point input.

The while-loop ensures that the program does this forever. Run the program and enter some numbers to try it out. Enter only positive integer values at first. Then, after a few successful tries, enter something that’s not a number (or not an integer). What happens?

The program crashes with the following error:

```
Enter a number:
9.2

Exception in thread "main" java.lang.NumberFormatException: For input string:
"9.2"
        at
java.lang.NumberFormatException.forInputString(NumberFormatException.java:65)
        at java.lang.Integer.parseInt(Integer.java:580)
```

```
        at java.lang.Integer.parseInt(Integer.java:615)
        at Crash2.doIt(Crash2.java:13)
        at Crash2.main(Crash2.java:22)
```

Notice, the error is occurring at line 13 of your program (your program might be slightly different depending on your typing). That is the line where you are trying to convert the string to an integer. The exception flagged, java.lang.NumberFormatException, is indicating that there was a problem in the number format: it wasn't a number that you entered.

To handle this, we need to extend Java's exception handling beyond that which we are already doing. Specifically, you need to add the following to you code:

```
catch (NumberFormatException nfe) {
       System.err.println("Problem: " + nfe);
}
```

Where do you put this?

Run the above program several times. Enter integer values, floating point values, as well as other garbage. Notice how your program is now robust enough to do the right thing when a number is input, and recovers in situations when nothing or some non-numeric input is provided. It does not crash!

Actually, the fact that the program does not crash causes a small problem. Namely, the program is still running. (It is not doing much, just waiting for you to provide input.). Figure out a way to kill your running program.