CSE12 Spring 2018
Homework/Programming Assignment #1
Due: Saturday, April 14, 11:59pm. (No LATE programs are accepted)
(100 points)

Goal:
For you to become comfortable in the lab using and using the unix command line, gain familiarity with turnin procedures, learn some basics of unit testing with JUnit, implement a few modest-sized programming problems using Arrays and LinkedLists (from the Java Collections Framework).

Logistics:
In EACH AND EVERY FILE that you turn in, we need the following in comments at the top of each file. These are essential so that we can more easily process your submissions and insure that you receive proper credit. This is a very large class with about 240 students when combining both discussion sections.

NAME: <your name>
ID: <your student ID>
LOGIN: <your class login>

Turn in:
We will have you turn in your code using Gradescope. We are still developing the procedures. A pinned Piazza post will describe the turnin procedures a few days before your program is due

There will be two components of turning in your code
1. Before you turn in, checking that your code will compile on a CSE Lab Machine
2. Using Gradescope to turn in a *copy* of your source code.

PLEASE NOTE: CODE THAT DOES NOT EVEN COMPILE WILL GET 0 CREDIT AND WILL NOT BE GRADED

While you may code/develop in any way you please, your code must work in the environment of the CSE Lab machines, which is standard CentOS 7 and Oracle Java.

→ **If you develop on Windows, you must convert your text files to be a Unix format before turning in cod**e.

You can turn in your assignment multiple times, but only the most recent one is recorded. Files are electronically time-stamped.

The files that will be collected:

Counter.java
CounterTest.java
ReverseArray.java
ReverseList.java

Getting Help in the Lab

All tutors, TAs, and the professor are here to help you, but we are not here to do your work for you. Here are some "must haves" for you to get help

- Your code must be properly indented. Tutors and TAs have been instructed to NOT even attempt to help you if your code isn't indented
- Your code must have sufficient white space to be readable. For example do NOT give us a code that looks like
  - `if(x==5&&(y+z)>=32||myVar){w=Math.sqrt(z*y+2*z);}`

  This is much more readable

  -
    ```
    if( x==5 && (y+z)>=32 || myVar ){
          w = Math.sqrt( (z*y) + (2*z) );
    }
    ```
- While you may use an IDE (integrated development environment) like Eclipse or Dr. Java, tutors/TAs/professor may not be able to help you if not familiar with your program. Here's a pro-tip, learn VIM and how to use the unix command line.
- You cannot monopolize a tutor's time. A tutor can tell you that "there are many people waiting for help, I can't spend any more time with you". Time is especially precious close to the due date/time.
- Tutors will NOT be in the lab after 10pm. That's on purpose and is an instruction by your professor. We will help you, but we will NOT help you push things until the last minute. Plan your time.

Getting Started

Create a subdirectory call "HW1" in your class account. All of your files should be placed in that subdirectory. If you cannot remember how to create directories, refer to a unix tutorial or reference sheet.

You will need to submit a pdf document named HW1-Answers.pdf. You can create this document using any text processing program that can generate a PDF file (Microsoft Office, OpenOffice, LaTeX, etc). Create this document now inside your HW1 directory, place the required comments at the top of the file, and save it as HW1-Answers (with the extension appropriate to the type of document you're currently working in--it will acquire the .pdf extension when you save it to PDF.

Problem #0 (5 points)

First read and sign the Integrity of Scholarship agreement for CSE 12 here:

Problem #1 (30 points)
The purpose of this problem is to get you more comfortable on the command line. For parts A and B, you may use any program you like to edit your java files (e.g., Dr. Java, vim, or even Eclipse), but we would like you to compile the program, run some junit tests, and generate Javadocs via the command line.

Download following Files from the Class Website and save them to your HW1 directory:

Counter.java
CounterTest.java
Counter.pdf
HW1-Answers.txt

A. Look at the file Counter.pdf. This is a PDF of documentation created using javadoc. Using whatever editor you like (vim, Dr. Java, or even Eclipse), modify Counter.java with appropriate javadoc comments, so that it generates similar documentation. Replace the author field with your name. By similar, we are not asking for exact wording, but ALL methods must be documented using javadoc

Next generate the javadocs for this file via the command line, and place all of the documentation files in a subdirectory called doc in your HW1 directory. If you do not know how to do this, and don't know where to start, try Googling "javadoc command line" (without the quotes). I recommend skipping the StackOverflow link and going to the official Java page. The section on "options" will be particularly useful.

You need to get comfortable with reading documentation to find out information for yourself. Any piazza posts in the vein of "how to do create javadoc…" will be removed by TAs/Tutors/Staff. It's a skill to read documentation on the web, and sorting out when you see conflicting "information."

Look at the generated Counter.html file to be sure it was generated appropriately, and matches what is in Counter.pdf (with your name as the author). When you turn in Counter.java, we will run javadoc on your file to create the required documentation.

In addition, place the following information in your HW1-Answers.pdf file

- What command line is used to create the javadoc documentation in HW1/doc?
- What command-line flag is used to to create the author and version entries for the class

B. Download the File CounterTest.java file. Most of the file is already complete, and you should be able to compile and run it. However, there are some 'TODO:' marked in comments where you are to complete the code. These completions including adding comments at the top of the file and completing the code to properly run some of the unit tests against the Counter class defined in part A.   When you run the unit tests, they should be meaningful tests and print out the following when running from the command prompt. A meaningful test is something that will verifies a particular input/output.

```
.Checking Default Counter Value is Zero
.Checking Proper Increment
.Checking Multiple Increments
.Checking Reset
.Checking Decrement

Time: 0.002

OK (5 tests)
```

1. The following are 4 possible ways to run the testing code from the command line, some work, some do not. **Note: this is JUnit 3, not JUnit 4**. In your HW1-Answers file tell us if the command line properly runs the code. If it does not run the code, briefly describe why. Feel free to use Google and any other internet site that helps you understand why some of these work and some of these fail.
    a. `java -cp '.:/usr/share/java/junit4.jar'  org.junit.runner.JUnitCore CounterTest`
    b. `java -cp '.:/usr/share/java/*'  org.junit.runner.JUnitCore CounterTest`
    c. `java -cp '.:/usr/share/java'  org.junit.runner.JUnitCore CounterTest`
    d. `java -cp '.:/usr/share/java/junit4.jar'  JUnitCore CounterTest`
    e. `java -cp '.:/usr/share/java/junit.jar'  JUnitCore CounterTest`

2. Modify Counter.java so that your Reset test fails. **The version of Counter.java that does not pass Reset test is the version you should turn in**. To be clear. Counter.java must *compile* but it should fail a reasonable Reset test. We will run your tests against an error-free version of Counter.java to insure that all tests pass. Then we will run your tests against your turned in version of Counter.java to see the failed Reset test.

In addition, place the following information in your HW1-Answers.pdf file. Again, feel free to look up these answers using Google or any other web resource.

- -cp and -classpath are “command-line switches” to the java and javac command to set Java’s classpath. What is another way to set the classpath without using a command-line switch? (hint: read http://docs.oracle.com/javase/tutorial/essential/environment/paths.html)

Problem #2 (40 points)

Create two java programs called ReverseArray.java and ReverseList.java. These programs are to provide the identical functionality, but using different implementations.

- Read a file of text *once,* line-by-line. Read each line of the file as a java String. The name of the file is specified as a command line argument.
- Print the file to standard output in reverse line order. That is, the last line of the file is printed first, next-to-last is printed next, and finally the first line is printed last (you do not reverse the text on each line)
- If the file does not exist, print "*File Not Found" to standard error* (it should use java Exception handling in a correct try..catch block. Nothing should be written to standard out in this case
- If a file is not supplied on the command line print a "usage" statement to standard error (Nothing should be printed to standard out)
- Your programs should generate no exceptions under (almost) any circumstances. Try to break with bad input. One type of input we will NOT test is giving your programs non-text files (also known as "binary files"). You do not need to check if a file is a text file.

If a file called manywords.txt exists, then to print the lines in reverse order, one would give the unix command

```
$ java ReverseArray manywords.txt
```

OR

```
$ java ReverseList manywords.txt
```

You can download two example files and outputs from the assignment folder

- short.txt, this is a 5 line text-based input file
- short.rev.txt, this is what the reversed output should look like
- declaration.txt, this is a text version of the US Declaration of Independence
- declaration.rev.txt this is the reversed version of the declaration. txt file

```
$ java ReverseList short.txt
```

(this should give you the identical file in short.txt.rev)

```
$ java ReverseList declaration.txt
```

(this should give you the identical file as declaration.rev.txt)

```
$ java ReverseList
usage: ReverseList <filename>
```

(the program prints this when no file is given)

you should test your program output for both ReverseList and ReverseArray

Using commands to validate your output is identical to the sample outputs.

In this course, you should become familiar with some unix command-line tools to validate that your code is doing what it should do.

1. Capturing standard output (stdout) of a program and saving to a file using the ">" (redirection) operator of the bash shell.
   First run your program and capture its output to a file.

   ```
   $ java ReverseList declaration.txt > myoutput.txt
   ```
   (this runs the java program ReverseList and puts the output in the file named "myoutput.txt")

2. Then, use the `diff` command to highlight any differences.

   ```
   $ diff declaration.rev.txt myoutput.xt
   ```
   (if there are no differences, then the diff command will not produce any output. That's what you want! If there are differences, diff will print out where it finds the problem. Any difference (even if you add a space at the beginning of a line, or an extra line at the end of your output) is notated by diff

   You can also view differences side-by-side, in the vi editor)

   ```
   $ vimdiff declaration.rev.txt myoutput.txt
   ```
   (This will open the vi editor and highlight the differences, if any)

1. Look up how to
   a. Redirect standard error (stderr) to a file in the bash shell
   b. Redirect stdout and stderr to the same file in the bash shell

<u>Specifics and Hints:</u>
Program organization: You will write the code in the main method of each class, though you might choose to implement helper methods to make the code simpler. When you create helper methods, make sure they are also declared as static.

In HW1-answers.pdf answer the following
- Why must the main method of java program be declared `static`

How are ReverseArray and ReverseList different? ReverseArray is implemented using an Array of Strings to store the file contents before printing (You may NOT use the ArrayList class or similar to implement ReverseArray, you MUST use normal arrays). ReverseList is implemented using a LinkedList of String type (i.e. LinkedList<String>) from the Java Collections Framework. We will be talking about Linked Lists and the Java Collections Framework in more detail in the next week or two, but for now we want you to use the documentation in the Javadocs API to figure out how to create and work with objects of type LinkedList<String>.

Implementation details for ReverseArray: You should initially create an array that can reference 100 Strings. If your file is longer than 100 lines (it WILL be when graded, so test this yourself!), when your program would read the 101st line, there isn't space to store it before printing. This should NOT cause an exception. Instead, create a new array of Strings with the ability to reference 200 Strings, copy the references from the old array into the new array and then continue reading. If file is longer than 200 lines, extend again by 100 slots in the array of Strings.

Commenting/Javadoc

1. Your ReverseList and ReverseArray programs must be commented in javadoc style.
2. Your code MUST BE INDENTED properly. You may use either spaces or tabs to indent your code, but please be consistent. If you use spaces, use at least 4 spaces to indent.
3. Other comments in your code are at your discretion. You should comment code if the comment aids in understanding how the code works. Too many comments is as bad as too few. In general, a short comment at the beginning of a block of code is all that is warranted. For example, you might comment a block of code with
   ```
   // Read file line-by-line and store in auto-expanding array
   ```
4. If we have not given hard and fast "rules" on how something should be done in your program/code, you aren't going to lose points for coding one way or another. There is more than one solution to any programming problem.

Problem #3 (25 points)
True/False. Create a text file (using a unix text editor like vi, or using a program like Microsoft Word and saving your file as a plain text (txt) file) called **Problem3.txt** and create answers with the number of the question followed by either the word True or False. One answer/line. eg.
1. True
2. False

and so on. This allows us to grade this part electronically.

This part is open book and open notes. You may use Google to help you determine the answers to these questions, and you may run any Java code to help you determine the answers. However, you may not ask your classmates for the answers nor may you give the answers to any of your classmates. The point is to *understand* the answers, as we assume that you have this knowledge from CSE 11 or CSE 8B and we will build on it.

1.	T	F	An instance variable declared as private can be seen only by the class in which it was declared and all its sub classes
2.	T	F	If a class C is declared as abstract, then private C myC = new C(); is valid.

3.	T	F	A variable declared as static cannot ever be modified, once it has been declared and initialized.
4.	T	F	The following is a legal statement: double x = 5;
5.	T	F	Code that does not explicitly handle checked exceptions, results in a compilation error.
6.	T	F	The declaration FilledOval[][] A = new FilledOval[20][30]; creates 600 FilledOval instances using the FilledOval() constuctor.
7.	T	F	A class that uses the Swing toolkit and wants to both display and be notified of changes to a JSlider must implement the ActionListener interface.
8.	T	F	method declarations void A(double x, integer k){}; and void A(integer k, double x) {}; have identical signatures
9.	T	F	The binary search algorithm will work properly on all integer arrays.
10.	T	F	if X is any valid, defined java object, then Object tmp=X; is a valid statement.
11.	T	F	`(“Give me Liberty”.split(“ “).length)` evaluates to 3
12.	T	F	`“”.format(“%d %s\n”,14, “shopping days left”);` is a valid statementt.
13.	T	F	If the following statements are the only two statements in a method, String X = “thing one”; and String Y = ”thing one”; then X.equals(Y) evaluates to true, but X == Y evaluates to false within that method.
14.	T	F	A class declared as final cannot be inherited via the extends keyword.
15.	T	F	consider the statement: String S = “Out of Gas”; then the statement: S[7] = 'g'; will change “Gas” to “gas”.
16.	T	F	boolean primitive variables can only be assigned values: true, false, or null.
17.	T	F	You can index into an array with a variable of type double as long as the there are no digits past the decimal point.
18.	T	F	The constructor of the super class is only called when the constructor of the sub class explicitly calls super(); as its first line.
19.	T	F	Any for loop can be rewritten using a while loop.
20.	T	F	It is legal to define more than one class in a java source file.
21.	T	F	A class can implement multiple interfaces
22.	T	F	`int i =  Math.sqrt(4.0);` is a valid statement.
23.	T	F	the `protected` keyword can only be applied to instance variables.
24.	T	F	Consider the statement: `throw new IllegalArgumentException();` This always causes the program to immediately exit.
25.	T	F	Suppose you have the following declaration: `int xyz = 4;` Then, in the body of a switch statement block

			`case xyz: System.out.println("4"); break;` is legal.