CSE12 - Fall 2017 HW #1

Warm up, and some fun with Rock, Paper, Scissors

(100 points)

Due 11:59pm Friday, October 6

Useful Resources:
Throughout this assignment, you may find the following resources helpful. Refer to them BEFORE posting questions on Piazza.

- Getting the starter code from Vocareum
- Submitting your assignment on Vocareum
- Connecting to the lab machines remotely
- Running bash on Windows
- Unix reference sheet
- Javadoc reference
- JUnit testing tutorial

Partner Policy:
You may work with a partner on this assignment using TRUE PAIR PROGRAMMING, as described in the Pair Programming Policy for the course. If you work with a partner, you will submit one set of code files and one PDF document with the written answers in parts 1 and 2. Howevever, part 3 must be completed *individually*.

Provided Files:
Counter.java
countertest/CounterTest.java
Counter.pdf
RockPaperScissors.java

Files to Submit:
countertest/Counter.java
countertest/CounterTest.java
HW1-Answers.pdf
RockPaperScissors.java

Goal:
For you to become comfortable in the lab using and integrated development environment (e.g. Eclipse) and 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, ArrayLists, LinkedLists and Iterators 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 400 students when combining both lectures.

NAME(S): <your (and your partner's) name>
ID(S): <your (and your partner's) student ID>
EMAIL(S): <your (and your partner's) email>

Turn in:
See the submission instructions in this document. You will need a Vocareum account for submission, so if you have not received your login information, post on the appropriate post on Piazza.

Getting Started
We strongly recommend that you work on the lab machines (in person or remotely--see instructions here), but if you choose to work on your own machine, you can. Just make sure your code runs correctly on Vocareum, as that is where we will be testing it.   Lab accounts will be set up sometime during week 1, hopefully no later than Wednesday. Instructions below assume you are using the lab machines.

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 program that can create pdf files (e.g., MS Word (export to PDF), LaTeX, Google Docs (export to PDF)). Create this document now inside your HW1 directory, place the required comments at the top of the file, and save it as HW1-Answers. **Remember, by the end of the assignment you'll need to save it as a PDF file in order to submit it.**

Problem #0 (5 points)

1. First read and sign the Integrity of Scholarship agreement for CSE 12 here:
   https://goo.gl/forms/0lJyCO3S2gHz8sgk1
   You cannot earn any credit in CSE 12 until you have done so.
2. Next (for 2 points) fill out a short pre-survey here:
   https://goo.gl/forms/gDthOq6mh8REkH5C2
3. For another 3 points, fill out a short pre-test.   Important:
   - You need your quia account to take this quiz
   - Once you start the quiz by clicking on the link below, you MUST finish it in one sitting. DO NOT use the back button, open a new tab or press enter. If you do so you will lose your one attempt at this quiz. (Post on Piazza to reset your attempt, but please try not to let this happen).
   - You will get your two points for any reasonable attempt at the quiz. However, do

your best because this is a good indicator of how prepared you are for CSE 12.
  - The secret word for the quiz is ‘letsdothis’ (without the quotes).
  - Finally, here’s the link: http://www.quia.com/quiz/6502945.html

Problem #1 (30 points)
The purpose of this problem is to get your comfortable both on the command line as well as using the Eclipse IDE. For parts A and B, you may use any program you like to edit your java files (e.g., Dr. Java, vim, Notepad++, or even Eclipse), but we would like you to compile the program, run some junit tests, and generate Javadocs via the command line. For part C, you will do the same steps in Eclipse.

Download following Files from Vocareum (see instructions here) or copy them from the public folder here **/home/linux/ieng6/cs12f/public/HW1/*** and save them to your HW1 directory:

Counter.java
countertest/CounterTest.java
Counter.pdf

Note that you should create a new directory inside your HW1 directory and place the CounterTest.java file there. Or you can just download the whole countertest folder, which contains the file CounterTest.java. In other words your directory structure should be the same as ours.

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.

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.

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(s) is/are used to to create the author and version entries for the class

B. Make sure you are done with part A before continuing. Next you will write JUnit tests and run them from the command line. If you are working on your own machine, follow the instructions

here to install **JUnit 4** (NOT JUnit 5) on your own machine. Note that these instructions omit the step where you need to add the hamcrest-core file to your CLASSPATH (you need to do this or you will get a compile error).
First, a little set up for these tests. We are using **JUnit 4** which requires that all code be in packages, so before we can run our tests, we need to move the Counter.java code into a package.

Move your Counter.java file into the countertest directory. Then, in a text editor, uncomment the line

```
package countertest;
```

at the top of the Counter.java file.

Now, open the CounterTest.java file in an editor. Most of this file is already complete, and you should be able to compile and run it (see below).   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 make reasonable tests and print out the following when running the textui-based TestRunner.

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

Time: 0.002

OK (5 tests)
```

Here’s what to do for this part:

1. First, complete the TODO items in CounterTest.java.
2. Next, run all of the tests from the command line. You will need to figure out the proper sequence of commands to do this.   In your HW1-Answers.pdf file, write the command that you used to successfully run the JUnit tests from the command line. Feel free to refer to the slides from class or to use Google for help.
3. Finally, modify Counter.java so that your Reset test fails. The version of Counter.java that does not pass the 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.

C. Next you will run the same tests, but in Eclipse. Follow the Eclipse documentation or any other tutorial to create a new Java project with the package countertest.   Create the class

Counter in the countertest package. You can load in the existing source code, or you can create a new class and copy and paste in the code from Counter.java. Then, create a JUnit test class called CounterTest that contains the code from CounterTest.java. Again, a simple way to do this is to create a new Test and then copy and paste the code in, or you can import the tester class source file. Use Google, talk to the tutors and talk to your classmates if you are having trouble. ***It is completely OK to help each other out with this part (i.e. getting set up to work in Eclipse) and is not considered cheating.*** Compile your code and run the JUnit tests again. Take a screenshot of your code loaded into Eclipse after running the tests. Place that screenshot in your HW1-Answers.pdf file.

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

- What was the most difficult/confusing part about getting set up in Eclipse?

For the rest of this assignment (and the rest of this course) you can use any editing environment you choose.

Problem #2 (40 points)
In this problem you will create a computer game to play the game of Rock-Paper-Scissors with a user. If you are unfamiliar with this game, you can read about it on Wikipedia here.

We have provided a tiny bit of starter code in the file RockPaperScissors.java. You will write your game in the main method. If you are using Eclipse, be sure this class stays in the default package. When the user starts your game, it should play the game of Rock Paper Scissors with the user until the user types ‘q’. Here is an example run. User input is shown in blue.

```
> java RockPaperScissors
Let's play!  What's your move? (r=rock, p=paper, s=scissors or q to
quit)
r
I choose rock. It's a tie
Let's play!  What's your move? (r=rock, p=paper, s=scissors or q to
quit)
p
I choose paper. It's a tie
Let's play!  What's your move? (r=rock, p=paper, s=scissors or q to
quit)
g
That is not a valid move.  Please try again.
(r=rock, p=paper, s=scissors or q to quit)
p
I choose rock. You win.
Let's play!  What's your move? (r=rock, p=paper, s=scissors or q to
quit)
s
I choose rock. I win!
```

```
Let's play!  What's your move? (r=rock, p=paper, s=scissors or q to
quit)
r
I choose rock. It's a tie
Let's play!  What's your move? (r=rock, p=paper, s=scissors or q to
quit)
s
I choose paper. You win.
Let's play!  What's your move? (r=rock, p=paper, s=scissors or q to
quit)
q
Thanks for playing!
Our most recent games (in reverse order) were:
Me: paper   You: scissors
Me: rock    You: rock
Me: rock    You: scissors
Me: rock    You: paper
Me: paper   You: paper
Me: rock    You: rock
Our overall stats are:
I won: 16%      You won 33%     We tied: 50%
```

Your exact formatting doesn't have to match ours, but the game play and which statistics are printed at the end should match. (Though of course, the exact values of the statistics will depend on the user's and system's moves in any given game!)

Here are some detailed requirements of the game play and specifics about the program:

- You will write your code in the main method, but you should use good style and helper functions as needed. Remember that these helper functions need to be static, because you are not creating any RockPaperScissors objects.
- The game should repeat until the user enters 'q'
- The game should track the full move history for both players. It should store the move history of the user in an array of Strings and the move history of the system in a LinkedList of Strings. These variables are already set up in the starter code. You just need to use them.
- The array that stores the user's moves is initialized to size 5. As the user enters more moves than the array will hold you should write code to expand this array (really, to make a new array and copy over the contents of the old one). The new array should always be twice the size of the old array. E.g. on the 6th move, the array becomes size 10, on the 11th move it becomes size 20, etc. You must do this resizing and copying "from scratch". That is, you have to write the code yourself, and **may NOT use any method in Java's Arrays class.**
- At the end of the game, the system should print out up to the *last 10* games, in reverse order. If there has not been 10 games, it should print out as many as has been played. If there have been more than 10 games, it should only print the most recent 10. (But

remember, it should store the full game histories). It should also print the win and tie statistics as in the example.

- Your program should gracefully handle incorrect input by re-prompting the user until they enter valid input. You can look for the letters r, s and p exactly, or allow more freedom in the input.
- Your programs should generate no exceptions under (almost) any circumstances. Try to break with bad input.
- It's up to you to decide how the computer player chooses its moves, but the one rule is *it cannot cheat.* That is, it cannot look at the user's move and then decide how to move. It can, however, look at the user's move *history* as well as its own history. You can do something as simple as having it choose a random move, or the same move, every time. But if you'd like to get more sophisticated (and possibly earn a **star point***), you should try to make the best player out there by taking into account what you know about how the user plays. The graders will play your programs, and we will feature the top performers in class. It's amazing how "smart" you can make your program with just a little information. **If you do get creative here, make sure you document your approach in your header comment at the top of your RockPaperScissors.java file.**
- Feel free to make a more creative version of RPS, for example, Rock Paper Scissors Lizard Spock is always fun. Get creative. Just make sure it's easy to understand how to play your game.

* **What the heck is a star point?** A star point is like extra credit, only it is not really factored into your grade. So why do the star point extensions?

1. The best reason to do these extensions is simply because you are excited about the problems. If you found the basic assignment interesting, but rather easy, and are hungry for more, then tackle these extensions!
2. Star points can also help your grade. If at the end of the term you have *almost* made it to the next higher grade and you've done enough star point extensions, you'll be pushed up a grade. What "almost" and "enough" mean cannot be determined in advance, so there's no way I can answer questions like "If I have a XX% and have done N star points, will I go up?"
3. Finally, star points communicate to me that you are deeply engaged in the material. So if at some point in the future you're looking for a letter of recommendation, or looking to be a tutor, star points give you something that I can brag about in your letter, or that make me believe you'll be a good tutor because you're deeply engaged with the material.

Problem #3 (25 points)
True/False. Take the following quiz: https://www.quia.com/quiz/6506535.html

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.

How to submit your homework

These are instructions for submitting all of your homeworks for CSE12.

You should have received an email from support@vocareum.com indicating your userid and password for this website. This is our submission site.

1, Follow the link https://labs.vocareum.com/home/login.php and log in.

2, You should be able to see HW1.

3, Click on the ”Upload” on the upper left corner. This assignment you have to create a new directory for countertest. Unfortunately, Vocareum does not support uploading folders. So you can create a folder and then upload files. **You should preserve the directory structure you used when testing your homework.**

4, We are setting the number of submission to be unlimited. So feel free to submit the assignment multiple times.

5, Once you submit the assignment, you should be able to view your submission report in Details->View submission report. We have created a check submission script for you to check if your submission has the correct directories, files and their structures. And all of codes compile.

You should be able to see that you pass all the checking points. Like the following:

```
---------------------------------------
Checking files
---------------------------------------
HW1-Answers.pdf file exists
---------------------------------------
Compiling files
---------------------------------------
RockPaperScissors.java file exists
RockPaperScissors.java compiles sucessfully
countertest/Counter.java file exists
countertest/Counter.java compiles sucessfully
countertest/CounterTest.java file exists
CounterTest.java compiles sucessfully
------ SUMMARY ------
All files exist
All files compile
```

8, Once you submit and check, you are done with submitting.

We are going to grade the latest version you submit. Passing the check points simply means that you don’t miss any necessary files and code compile. However, you should still keep testing your code.