PSA 2: Turtle Loops and Sleepwalking student

Read all instructions in this document before starting any coding!

Due: Sunday, October 22th, 11:59pm

In this assignment you will program a turtle taking a random walk through its world. While it's fun to see what path a sleepy Turtle takes as it wanders about, random walks are also important in the scientific world. The Wikipedia page on Random Walks has a nice overview of how they are used in many different fields. Your Turtle will take a 2-dimensional random walk. That is, it will move from the bottom to the top of the screen, randomly moving either a bit to the left or a bit to the right with each "step" upward…

As described in class, **you may complete at most 4 of the quarter's programming assignment using pair programming.** If you have decided to work in pairs, please review the guidelines for pair programming. In addition, note the following details about working with a partner:

- For this assignment, if you decided to work in pairs, find a partner before you start on the assignment. Or else, you have violated the policy on pair programming.
- You will submit only ONE version between the two of you though your group can submit as many times as you want to.

Starter code: LINK HERE

Helpful Information:

Online Communication: Using Piazza, Opening Regrade Requests

Getting Help from Tutor, TA, and Professor's Hours

Lab and Office Hours (Always refer to this calendar before posting.)

Academic Integrity: What You Can and Can't Do in CSE 8A

Remote Lab Access and File Transfer Guide or **Yingjun's SSH and VNC post**

Turtle Documentation

Table of Contents:

Getting Started

Instructions for working on a B230 lab machine:

1) Open a terminal
2) Copy the starter code from the public folder.

Paste the following lines of code one after the other in the terminal i.e. paste the first line, press enter and then the second line and so on:

```
cd ~/
mkdir psa2
cd psa2
cp ../../public/psa2/*.java .
ls
```

(Do you recall what each of the above commands mean. If you are not able to, review PSA0, where each of the commands is explained.)
3) Verify that Turtle.java and RandomTurtleTester.java is shown as the result of the last command.
Following these procedures, your file will be named correctly and be located in the correct place. For every assignment you need to follow correct naming assignments in the correct locations to get credit for your work.
4) Proceed to start your programming assignment.

Instructions for working on your Local Machine (Your Laptop/Computer):

1) Create a psa2 folder on your machine.
2) Save Turtle.java and RandomTurtleTester.java inside the psa2 folder: LINK HERE

Problem

Part 1: Taking a Random Step (3 points)

Your first task is to write a method `generateOffset(Random generator)` in Turtle.java: (.75 point)

```
  /** Determine how large of a step the turtle takes
    * Parameters: generator: a Random object that will help to generate random number.
    * Return value: A random number between 1 and 3.
    * */
 public int generateOffset(Random generator)
```

This helper method should generate a random number between 1 and 3, inclusive, and the amount the Turtle moves during a step will be proportional to what this method returns.

Like in PSA1, a Random object is passed into the method to generate a random number, instead of creating a new Random object each time the method is called. A single Random object will be shared throughout each of your programs. This is because as a computer scientist, it is your job to ensure that not only your program will run and give correct output but also your code be efficient. One of the most important aspect of clean coding is to not repeat yourself (the DRY principle in software engineering). It means that you should not create duplicates if they all have the same functionality.

Then write a method `takeStep(int direction, Random generator)` in Turtle.java: (1.5 points)

```
  /** Display a random step of the turtle.
    * Parameters:  direction: -1 for step to the left or 1 for step to the right.
    *              generator: a Random object that will help to generate random number.
    * Return value: How large of a step (the offset 1-3)
    * */
 public int takeStep( int direction, Random generator)
```

This method should take an integer, and it should move the Turtle a bit in the specified direction. The `takeStep(...)` method should call a helper method called `generateOffset(...)` to determine how large of a step the Turtle takes. You should assume the Turtle will always be facing up when this method is called. **This method should also be sure to rotate the Turtle to facing up again before it ends.**

One way to do this is to turn the Turtle, say, 45 degrees to the left or the right, depending on the value of direction. Hint: Use the `turn` method, and not `turnRight` or `turnLeft`. Then move the Turtle forward some small amount depending on the offset. For example, if you use 20 as that small amount (or a value of your choice): the Turtle could move 20, 40, or 60 depending on the offset in a direction.

Test your method! (.75 point)

When you are done with this method you should test it. You will do this by writing some code in the main method of the `RandomTurtleTester` class. This is the class you will run to test all of your methods.

Although you will eventually want your Turtle to take random directions for its steps, for now in your main method you should:

1. Create a new World
2. Create a new Turtle at the bottom center of this world.
3. Call `takeStep` with the argument 1 to test taking a step to the right.
4. Call `takeStep` with the argument -1 to test taking a step back to the left.

When you have done all of the above, you should see your Turtle take a little step to the right and then back to the left as it moves a short distance up the page, as in the following screenshot. Make sure to use comments above the block of tester code to indicate that it is for the method(s) you are testing. **Your tester will not be given points if they are not commented properly.** Keep in mind that the size of the step can vary depending on the offset. **Notice that the Turtle ends still facing upward.**

The method `takeStep` returns 1 and then 2 in the example above.

Part 2: Taking a Random Walk and Calculating Displacement (5 points)

For the next part you will repeatedly call your `takeStep` method, passing it in a random value for the direction, to simulate a random walk. To do this, we will need to write a couple more methods (one is already provided to you).

Back in your Turtle class (in `Turtle.java`) write a method :

`public int rwPosition(int nSteps, Random generator)`

which takes as input the number of random walk steps that the calling object Turtle will take and a Random object.

This method will do the following:

1. Make the Turtle take `nSteps` steps, randomly. Making the Turtle take each step will involve first a call to `getRandomStep` (a method we provide, explained below) and then a call to `takeStep`, using the return value of `getRandomStep` as input to `takeStep`. **(2 points)**
2. Count and return the Turtle's signed final displacement in terms of steps **horizontally to the left or to the right of the Turtle's original position** (this is only ONE dimension, we are not counting movement up or down!). For example, **if nSteps is 5 and the Turtle chooses to take the following steps: -3, -1, 3, 1, -1 (negative means going left), then the method would return -1**. Notice that this value might be different every time the method is called. **(2 points)**

Hint: You will need to use a loop in this method, and a variable to keep track of the signed displacement each time the Turtle takes a step. To get the displacement, you might want to review what `takeStep` returns. You will update this variable inside the loop. Either a for-loop or a while-loop will work equally well for this part.

`getRandomStep()` method provided `Turtle.java` file:

```
    /**
     * Return whether or not the turtle should take a step to the left
     * or the right
     * Parameters: probabilityToRight: the probability the turtle moves to the right.
     *                                 Should between 0-1.
     *             generator: a Random object that will help to generate random number.
     * Output: -1 to represent a step to the left, 1 to represent a step to the right.
     */
    public int getRandomStep(double probabilityToRight, Random generator) {
        // Choose either 0 or 1 (note that next int is NON inclusive
        // of its argument).
        double choice = generator.nextDouble();
        if (choice < probabilityToRight) {
```

```
            return 1;
        } else {
            return -1;
        }
    }
```

The `getRandomStep(double probabilityToRight, Random generator)` method takes two arguments, and returns either -1 or +1, indicating the direction the Turtle will stray on its next step. -1 means the Turtle should move a bit to the left, while +1 means the Turtle should move a bit to the right.

By default, we want the probability of moving left and moving right the same. So you should pass 0.5 to `getRandomStep()`.

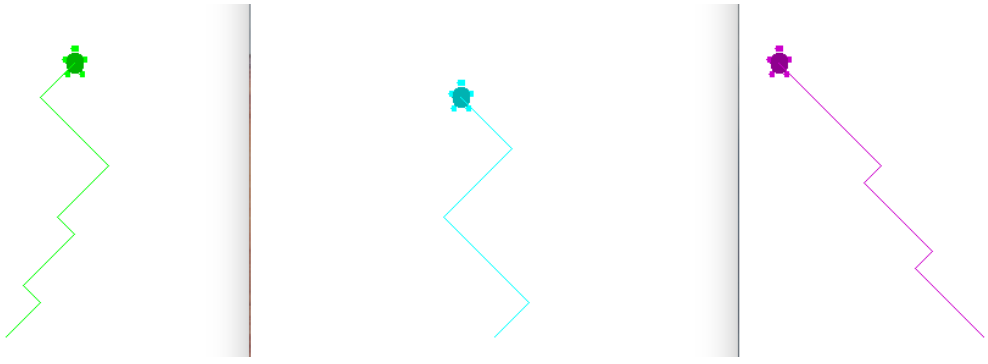
Next, you will write code to test `rwPosition` method. Again, back in the main method of `RandomTurtleTester` *add* code to do the following:

You should NOT remove the tests from Part 1. **(1 point)**

1. Create a NEW World object.
2. Create a NEW Turtle at the bottom, center of this World.
3. Call your `rwPosition` method just once, passing in a small number for `nSteps` (e.g. somewhere between 5-10 is probably a good number).
4. Print out the final signed displacement of the Turtle in the main method

Now, to test your method *run your program multiple times.* Each time you run it you should verify that your Turtle moves smoothly and seemingly randomly up the page. The path should probably be different every time. You should also verify that the Turtle's final position matches the signed displacement returned by your method (and printed in your main method). Remember, you can always use System.out.println to ensure that your displacement calculation is correct. Make sure to use comments above the block of tester code to indicate that it is for the method(s) you are testing. **Your tester will not be given points if they are not commented properly.**

Here are some example runs: Three runs of the same program give different results because turtle walks randomly.



The method returns 4, -2 and -12 respectively in above executions.

Part 3: Counting Steps given Displacement (6 points)

Next, back in your Turtle class, add a method :

`public int countSteps(int maxDisplacement, Random generator)`

which takes as input the **maximum unsigned displacement** (maximum displacement in either direction) that a Turtle is allowed and a Random object. That is, `maxDisplacement` is the amount that a Turtle is ever allowed to stray in one direction or another. The Turtle should stop when it has gone farther than `maxDisplacement`. This method will do the following:

1. Display the Turtle's randomly walking up the page, *until it reaches* `maxDisplacement` distance from its starting position, either to the left or the right. **(2 points)**
2. Count and return the number of steps the Turtle took before it hit the maximum allowed displacement. Note that no matter how long a step is, it is still considered a single step. For example, a step of distance 3 is still one step. **(2 points)**

You'll again need a **loop**, as in the previous part, but this time your loop condition will be concerned with the turtle's displacement from its starting position. Notice that if the Turtle takes a step to the left with offset 1, its displacement is now -1, but if it then takes a step to the right with offset 1, its displacement goes back to 0. So if its maximum allowed displacement is 2, it has not yet hit it. It might wander for quite awhile before hitting its maximum displacement, and **it's OK if the Turtle hits the top of the screen or it happens to exceed the maximum displacement due to taking a large last step.**

Some useful tips: How would you check whether the turtle has reached `maxDisplacement`? If `maxDisplacement` is 5 and turtle's signed final displacement is -5 then the turtle has reached 5 steps of displacement (to the left) and should stop moving. Since a turtle's final displacement could be negative, you will need to check whether the ***absolute value*** of a turtle's displacement has exceeded the `maxDisplacement`.

You can use the existing Java method Math.abs() to calculate the absolute value:

`Math.abs(-1)` gives 1

`Math.abs(1)` also gives 1

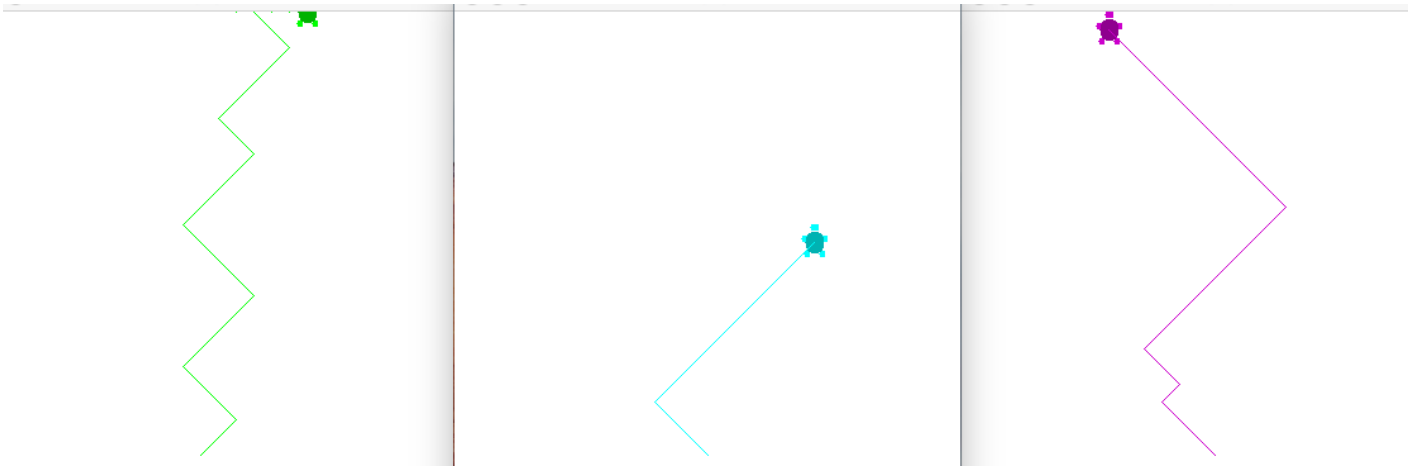
Again you will test this method by adding code to do the following in your `RandomTurtleTester`'s `main` method: **(2 points)**

1. Create a NEW World object.
2. Create a NEW Turtle at the bottom, center of this World
3. Call your `countSteps` method just once, passing in a small number for `maxDisplacement` (again, somewhere between 5-10 is probably a good number).
4. Print out the number of steps the Turtle took to reach the maximum displacement.

Again, to test your method *run your program multiple times.* Each time you run it you should verify that your Turtle moves smoothly and seemingly randomly up the page. The path should probably be different every time. You should also verify that the Turtle's final position stops after it matches or exceeds (in the

scenario that the last step is larger than 2) the displacement allowed, and that the Turtle actually took the number of steps that are returned.

Here are some example runs with max displacement of 6:



The method returns 24, 7, and 15 respectively in above executions.

Part 4: Simulation & Write up (3 points)

In this part, you will use the code you wrote above to do some science/analysis on random walks.

Getting Started:

1. Save a second copy of your `RandomTurtleTester.java` file as `RandomTurtleTester2.java`, be sure to update the name of the class in this new file to **`RandomTurtleTester2.`** Use this file to perform the tests necessary to answer the questions below.

2. Place the following comment at the beginning of your RandomTurtleTester2.java file. This is where you will later fill in your answers to questions A and B given below.

```
/* Filename: RandomTurtleTester2.java
 * Created by: CSE 8A Staff, cs8af
 * Date: Fall 2017
 *
 *  A)
 *
 *  B)
 *
 */
```

A. What is the average final signed-displacement for a random walker after making 100 random steps with different values for the **probabilitytoRight** parameter (the parameter passed to `getRandomStep` in `rwPositionPlain`)? You should at least try **three different values for probabilitytoRight**, one should between **0-0.5**, one should be **0.5** and the last one should between **0.5-1**. The average is of the total displacements after many trials, see sample code below for clarification.

B. Describe the results in question A when you vary the **number of steps** and **probabilitytoRight** (take 2 other values for number of random steps instead of 100).

To answer questions A and B, you should adapt the random-walk method you wrote to investigate these questions. In particular, you should:

Write the required code to answer questions A and B above **(1 point)**

Writing the Code:

To-do item #1 Write another method **`rwPositionPlain`** in `Turtle.java` that does not move the Turtle (call `generateOffset` instead of `takeStep`). It should simply return the final displacement, in the same way `rwPosition` did.

To-do item #2 Come up with a plan for how you will answer questions A and B. This plan should incorporate the ideas demonstrated in the following codes:

```
World w = new World();
// It is not important where the Turtle starts, because you won't actually move
the Turtle
Turtle t = new Turtle( w );
Random generator = new Random();
int numTrials = 0;
double totalDisplacement = 0.0;
while ( numTrials < 142 )
{
  totalDisplacement = totalDisplacement + t.rwPositionPlain( 100, generator );
  numTrials = numTrials + 1;
}
// Find the average here....
```

Not surprisingly, the 142 is not important -- except when you want to find the average of the values created! It is simply used as a number that indicates that you've done "a lot" of different trials so you can take their average.

To-do item #3 Implement the above code in your `RandomTurtleTester2` main method. Print out the result of the final average.

To-do item #4 Do some experiments by changing the value 100 to other values of `nSteps`, and see how this affects your final average.

Once you've finished writing the required code, you should then run it and use the results to answer questions A and B above.

Fill the answers to A and B in the header of your file `RandomTurtleTester2.java` **(2 points).**

Code Style and Comments (1 point)

Remember to properly comment your code, use meaningful variable names and **place both partners' (if you work in pairs for this PSA) names, logins, and the date in the header of ALL YOUR FILES.**

Program Description (2 points)

Describe what your program does as if it was intended for a 5 year old or your grandmother. Do not assume your reader is a computer science major. The programs you should comment on in this segment include **RandomTurtleTester, RandomTurtleTester2 and Turtle.java. Write this as comments at top of Turtle.java file.**

Star point (Optional)

The Star point option this week is to investigate another simulation problem using randomness (also called Monte Carlo experiments). **New for this week: you may complete the Star point with your PSA2 partner or on your own, but you must place a header comment at the top of your file to make it very clear whose work it is.** For this Star point, you should create a NEW Java class called `StarpointSimulation` and place your code in a main method in that class. **Don't forget your header comment in that file which includes all the usual information, and especially whose work this is and who should be given Star point.**

The Star point is open ended, as usual, but to earn this point you must do the following at a minimum:

1. Choose a problem which can be investigated via simulation using randomness. This problem should be non-trivial (e.g. flipping a fair coin is not complex enough.) Possible problems you might consider include:
    1. Rolling (loaded) dice -- simulate rolling a pair of dice, first fair, and then loaded (i.e. dice that are weighted in some way), and compare your results to what you would predict theoretically (this requires some knowledge of basic probability, but feel free to look up probabilistic expectations for dice rolling).
    2. The Monty Hall problem -- show via simulation why you should always switch your guess
    3. Blackjack -- create a simulation to show why the house always wins against a person playing with different strategies.
2. Create a plan as to how you will investigate your hypothesis. Write the code to implement your plan.
3. Run your experiments.
4. Write up your experiments and your results in a comment in your `StarpointSimulation.java` file. Include the following:
    1. What problem did you investigate?
    2. How did you design your investigation? What was your experiment? How did the code you wrote allow you to do the experiment?
    3. What do your results show?

This Star point option is **NOT** for the faint of heart. It can be pretty interesting, but will definitely take some time and independent research.

How to Turn in your Homework

1. Place your psa2 folder in the home directory of your student account (cs8afxx).

2. Ensure that the psa2 folder contains at least all the following files: **`RandomTurtleTester.java`, `RandomTurtleTester2.java`, and `Turtle.java`. If you did the Star point you will also need to turn in StarpointSimulation.java**

3. Run the command: `cse8aturnin psa2`

4. Follow the prompts.

5. Verify that your homework was turned in with the command: `cse8averify psa2`

Refer back to psa0 turnin instructions for additional details.