CSE 142, Autumn 2007
Programming Assignment #5: Random Walk (20 points)
Due: Tuesday, October 30, 2007, 4:00 PM

Program Description:

This assignment focuses on `while` loops, random numbers, and using objects. Turn in a file named `RandomWalk.java`. The program draws a pixel-sized "random walk" that moves in random directions on a `DrawingPanel` until it has moved a certain distance away from its starting location. A random walk is a visualization of the idea of taking repeated small steps in random directions. Random walks can be done in one, two, or more dimensions. Read more about interesting mathematical properties of random walks here:

- http://en.wikipedia.org/wiki/Random_walk

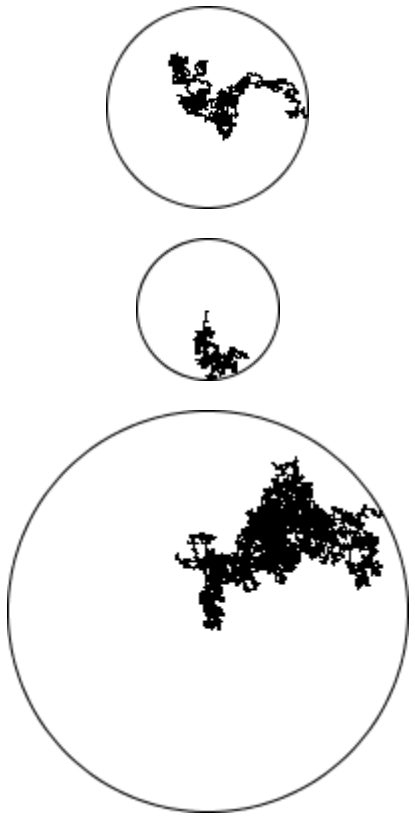
```
<< your introduction message here >>

Radius? 50
I escaped in 2468 move(s).
Walk again (yes/no)? y

Radius? 35
I escaped in 987 move(s).
Walk again (yes/no)? YES

Radius? 100
I escaped in 13713 move(s).
Walk again (yes/no)? n

Total walks = 3
Total steps = 17168
Best walk   = 987
```



The program begins with an introduction of your choice. Print anything you like, but keep it to a reasonable number of lines, no offensive remarks, reading a `Scanner`, infinite loops, etc.

Your program should perform 1 or more random walks. A walk begins by asking the user for the radius (in pixels) of the walk area. After the user inputs a radius, a `DrawingPanel` appears with a white background and a black-outlined circle with this radius. The panel's size should be exactly enough to contain this circle: one pixel more than twice the radius. For example, a circle of radius 100 should be drawn in a `DrawingPanel` of size 201.

Next the `DrawingPanel` animates the steps of the random walk. The walker is a single black pixel that begins in the center of the circle. Every **5ms**, the walker randomly moves its position up, down, left, or right 1 pixel. The walker should choose between these four choices randomly with equal probability. The walk ends when the walker reaches the perimeter of the circle (when it walks so far that its direct distance from the center is greater than or equal to the circle's radius). When the walk ends, the program reports how many moves were made.

After each game, the program asks the user to do another walk. Assume that the user will give a one-word answer. The program should walk again if the user's response begins with the letter **Y**. That is, answers such as `"y"`, `"Y"`, `"YES"`, `"yes"`, `"Yes"`, or `"yeehaw"` all indicate that the user wants another walk. Otherwise assume that the user does not want to walk again. For example, responses such as `"n"`, `"N"`, `"no"`, `"okay"`, `"0"`, and `"hello"` would mean that the user doesn't want any more walks.

If the user wants to do another walk, the steps described above repeat, including a new `DrawingPanel` for the new walk. Each random walk occurs in its own properly-sized `DrawingPanel`.

If the user chooses not to walk again, the program prints overall statistics. The total runs, total moves for all runs, and the best run (the one that required the fewest moves) are displayed.

The log on this page demonstrates your program's behavior. Your program will generate different random moves, but your output structure should match this exactly. If you like, you may assume that no run will require more than 999,999,999 moves.

Implementation Guidelines:

The repetition in this program should be done using `while` loops. We suggest that you develop this program in stages:

- Work on a text version of the program before adding the graphics.
- Initially write a version that does just one random walk, not many walks.
- Initially test using very small radius values such as 3 or 5.
- Put in temporary "debugging" code to print out the walker's initial position and position after each move.

The following might be a log of execution for a preliminary version of your program in progress. (You do not *have* to develop the program this way and should not turn in a final program that prints each move):

```
Radius? 3
- at the start: x=3, y=3
- after moving: x=3, y=2
- after moving: x=2, y=2
- after moving: x=2, y=1
- after moving: x=1, y=1
- after moving: x=1, y=2
- after moving: x=1, y=1
- after moving: x=0, y=1
I escaped in 7 move(s).
```

The way to tell whether the walker has exited the circle is by examining distances between points. The formula to compute the distance between two points is to take the square root of the squares of the differences in x and y between the two points. For example, the distance between the points (11, 4) and (5, 7) is $\sqrt{(11-5)^2+(4-7)^2}$ or roughly 6.71. However, if you write this program the way we intend, you shouldn't have to manually program this formula yourself.

Represent the random walker's initial position and current position as `Point` objects (as seen in Chapter 3) and use those objects' methods for any relevant computational tasks. Remember to `import java.awt.*;`

Produce randomness using a single `Random` object. `Random` objects produce random integers. These can be mapped to arbitrary random choices. For example, to randomly choose a color between red, yellow, and blue, pick a random integer from 0 through 2, and consider 0 to be red, 1 to be yellow, and 2 to be blue. Remember to `import java.util.*;`

Note that you shouldn't construct the `DrawingPanel` for each run until you have read the radius from the user. You can cause your `DrawingPanel` to pause for 5ms by calling its `sleep` method with a parameter value of 5. Doing so repeatedly between drawing operations causes the appearance of animation.

Assume valid user input. When the user is prompted for numbers, the user will type valid integers in proper ranges. When the user is prompted to play again, the user will type a one-word string as their answer. To deal with the yes/no response from the user, you may want to use some of the `String` class methods described in Chapters 3 and 4 of the book.

Stylistic Guidelines:

Structure your solution using static methods that accept parameters and return values where appropriate. For full credit, you must have at least **3 non-trivial methods other than main** in your program. Two of these must be the following:

- a method to perform a **single "random walk"** on a `DrawingPanel` (not multiple walks)
- a method to **report the overall statistics** to the user

You may define other methods if they are useful for structure or to eliminate redundancy. If you like, you can place the loop that performs multiple walks, and the prompt to the user to walk again, in `main`. As a reference, our solution has 3 methods other than `main` and occupies 80 lines total, though you do not need to exactly match this.

For this assignment you are limited to the language features in Chapters 1 through 5 of the textbook. Use whitespace and properly. Give meaningful names to methods and variables, and follow Java's naming standards as specified in Chapter 1 of the textbook. Localize variables whenever possible. Include a comment at the beginning of your program with basic description information and a comment at the start of each method. Since this program has longer methods than past programs, also put brief comments inside the methods explaining the more complex sections of your code.