

**READ ALL INSTRUCTIONS BEFORE STARTING YOUR SOLUTION.**

**Topics:** ch. 5: while loops, more analysis, structured, top-down **design**, randomization, etc.

**Specifications:** (3-13 should be in style guide)

1. Create a program named Sp7yourInitials\_Walk (e.g. Ed Bob Hyde would use Sp7EBH\_Walk.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. You may read about mathematical properties of random walks: [http://en.wikipedia.org/wiki/Random\\_walk](http://en.wikipedia.org/wiki/Random_walk)
2. You must meet all Java\_Program\_Style\_Requirements\_GradingComponents by due dates/times for full credit.
3. At the top of your console output, display author, title and a brief introduction; but, no offensive remarks.
4. The main method must clearly show all major activities- the overall program structure, not header/body/trailer.
5. Where a program repeatedly gets user input and displays results, the main code must indicate that clearly.
6. The main method must not read input, nor print nor draw output, itself; it must delegate that work.
7. You must have one method to do console output and a separate method to do any required graphical output.
8. The code that asks the user for input must not be in the same method as any code to read data from a file.
9. You must put any data files in the same directory as your program.
10. For graphic output, DrawingPanel.java in the same directory as your program.
11. You must structure your solution using static methods that accept parameters and return values where needed.
12. You must have **at least 3 non-trivial methods other than main** in your program – see the next specification.
13. You should define other methods as needed for structure or to eliminate redundancy.

**Sample console interaction:**

<< *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

**Specifications:**

14. Two of your **non-trivial methods** must be the following:

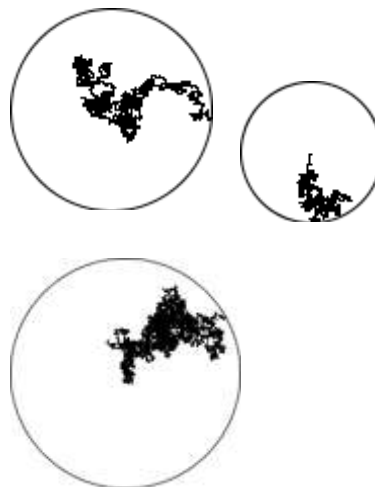
- I. a method to perform a **single "random walk"** on a DrawingPanel (not multiple walks)
- II. a method to **report the overall statistics** to the user

15. Your program must perform 1 or more random walks each time it is run. You may assume each step in a walk is 1 pixel.

16. A walk begins by asking the user for the radius (in pixels) of the walk area. (Validate input.)

17. 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 just large 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 or 202.

18. Next use DrawingPanel to animate the steps of the random walk that starts at the center of the circle. The walker can be a single black pixel that begins in the center of the circle. Every few milliseconds (5 ms for example), the walker randomly moves its position up, down, left, or right 1 step. The walker must choose

**Sample DrawingPanels:**

between these four directions **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 or steps were made to get out of the circle. Your code may **only create one random object per run**.

19. 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. What should you do for an empty input string?
20. If the user requests another walk, the steps described above repeat. Each random walk occurs in its own properly-sized DrawingPanel. You do not need to have your program close the earlier DrawingPanels.
21. If the user chooses not to walk again, the program prints overall statistics. The **total walks**, **total moves** for all walks, and the **shortest walk** (the one that required the fewest moves) are displayed.
22. The samples above demonstrate a program's behavior. Your program will generate different random moves, but your output structure should match this exactly. You may set a maximum of 999,999,999 steps.
23. Develop this program in stages:
  - a. First design and develop a text version of the program before adding the graphics.
  - b. Initially write a version that does just one random walk, not many walks.
  - c. Initially test with a small, easily visible 5 pixel rectangle rather than a single pixel.
  - d. Put in temporary "debugging" code to print walker's position after each move.

The following might be a log of execution for a preliminary version of the program:

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).
```

24. One way to tell whether the walker has exited the circle is by examining distances between points. A formula to compute the distance between two points would 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  $\sqrt{(11-5)^2 + (7-4)^2}$  or roughly 6.71. However, **you may not program this formula yourself**. You must **import java.awt.\* and use the Point class** (ch. 8, p. 504, p. 1126) to represent the random walker's initial and current position, and use Point objects' methods for relevant computational tasks **such as calculating distance**.
25. You must 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 from among red, yellow, and blue, pick a random integer from 0 through 3, and consider 0 to be red, 1 to be yellow, 2 to be blue and 3 to be green. Remember to import java.util.\*.
26. You must not construct the DrawingPanel for each run **until you have read the radius from the user**. You can cause your DrawingPanel to pause for about 5-25 ms by calling its sleep method with a argument value of 5. Doing so repeatedly between drawing operations causes the appearance of animation. You should experiment with the sleep time.
27. You may partially assume valid user input. When the user is prompted for numbers, assume the user will type valid integers in proper ranges. When the user is prompted to play again, assume the user will type a one-word string as the answer. To deal with the yes/no response from the user, use some of the String class methods described in our class and in Chapters 3 and 4 of the book.