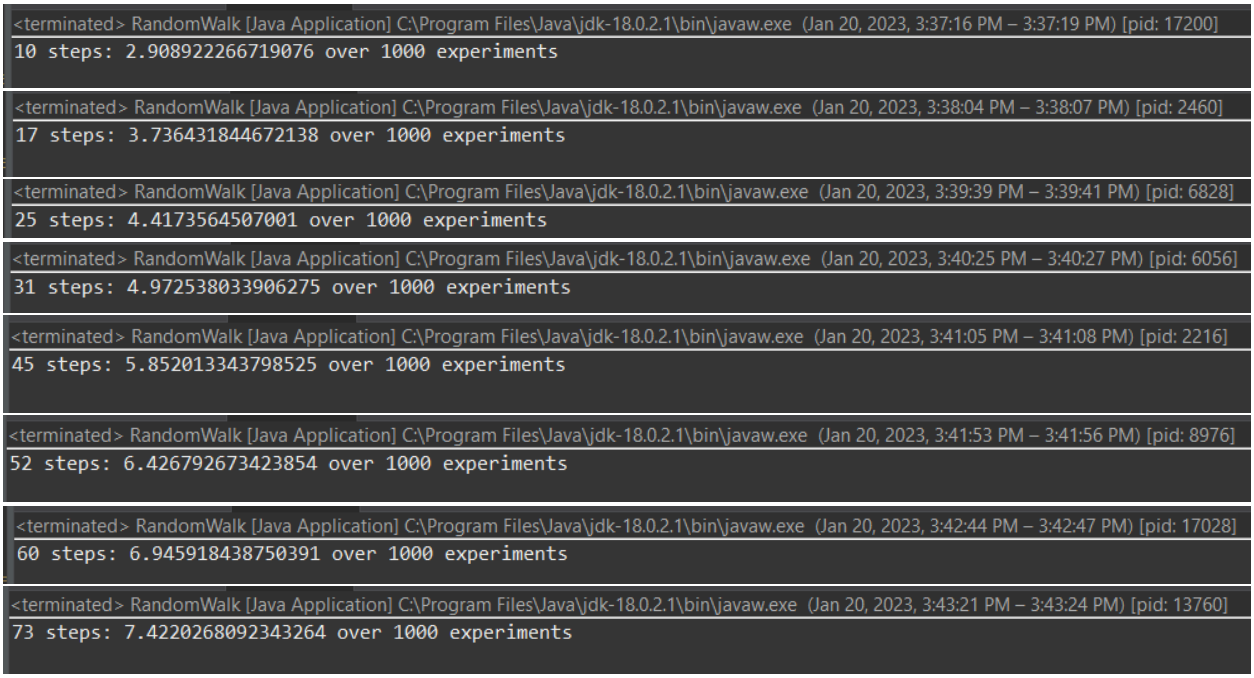
Assignment 1

The following table consists of the mean distance calculated for different values of m.

Steps (m)	Distance (d)
10	2.909
17	3.736
25	4.417
31	4.973
45	5.852
52	6.427
60	6.946
73	7.422
82	8.142
96	8.772

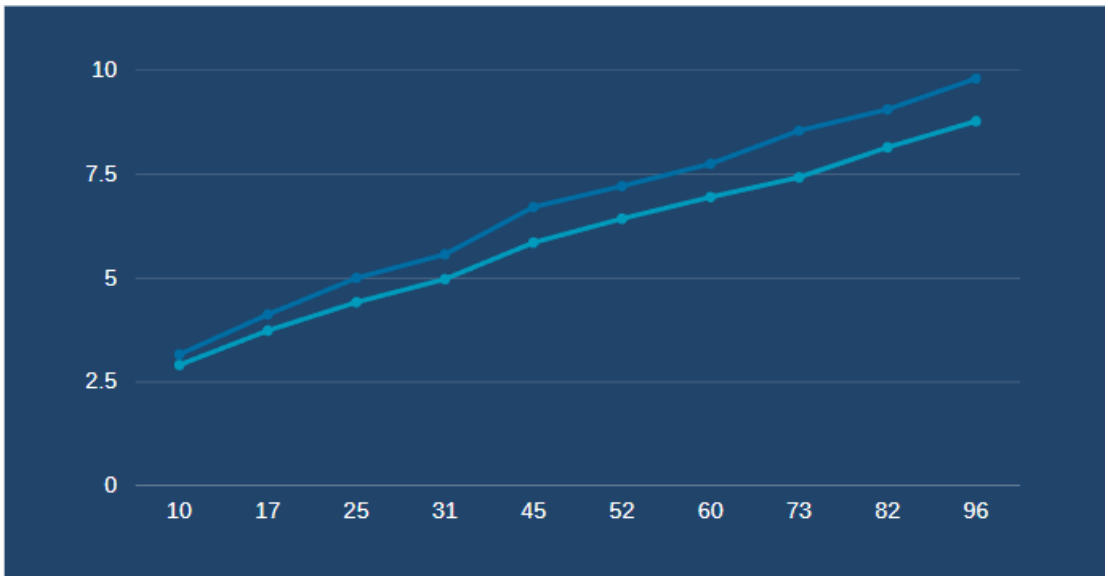
Here are the screenshots of the mean distance (taken over n=1000) calculated by the code:



```
<terminated> RandomWalk [Java Application] C:\Program Files\Java\jdk-18.0.2.1\bin\javaw.exe  (Jan 20, 2023, 3:44:04 PM – 3:44:06 PM) [pid: 2920]
82 steps: 8.141741949096446 over 1000 experiments

<terminated> RandomWalk [Java Application] C:\Program Files\Java\jdk-18.0.2.1\bin\javaw.exe  (Jan 20, 2023, 3:44:37 PM – 3:44:40 PM) [pid: 16424]
96 steps: 8.771632354423778 over 1000 experiments
```

The trend appears to be in the direction of the square root of the mean distances. The following table and graph show the relationship:

Steps	Distance	Square root of steps
10	2.909	3.162
17	3.736	4.123
25	4.417	5
31	4.973	5.568
45	5.852	6.708
52	6.427	7.211
60	6.946	7.746
73	7.422	8.544
82	8.142	9.055
96	8.772	9.798



By this, we can determine that d is directly proportional to the square root of m.

d α $\sqrt{m}$

Which means that:

d = k$\sqrt{m}$

To find out the value of k, I found out the value of **d / $\sqrt{m}$** as shown in the following table:

Steps (m)	Distance (d)	Square root of steps ($\sqrt{m}$)	k (d / $\sqrt{m}$)
10	2.909	3.162	0.9199873498
17	3.736	4.123	0.9061363085
25	4.417	5	0.8834
31	4.973	5.568	0.8931393678
45	5.852	6.708	0.8723911747
52	6.427	7.211	0.8912772154
60	6.946	7.746	0.8967208882
73	7.422	8.544	0.8686797753
82	8.142	9.055	0.8991717283
96	8.772	9.798	0.895284752

To get a generalized approximate value of k, I take the average of all the above calculated values of k. It comes out to be **0.893**

Thus, we can say that k is approximately equal to 0.893

So, we get the equation as: **d = 0.893 $\sqrt{m}$**

Code:

RandomWalk.java

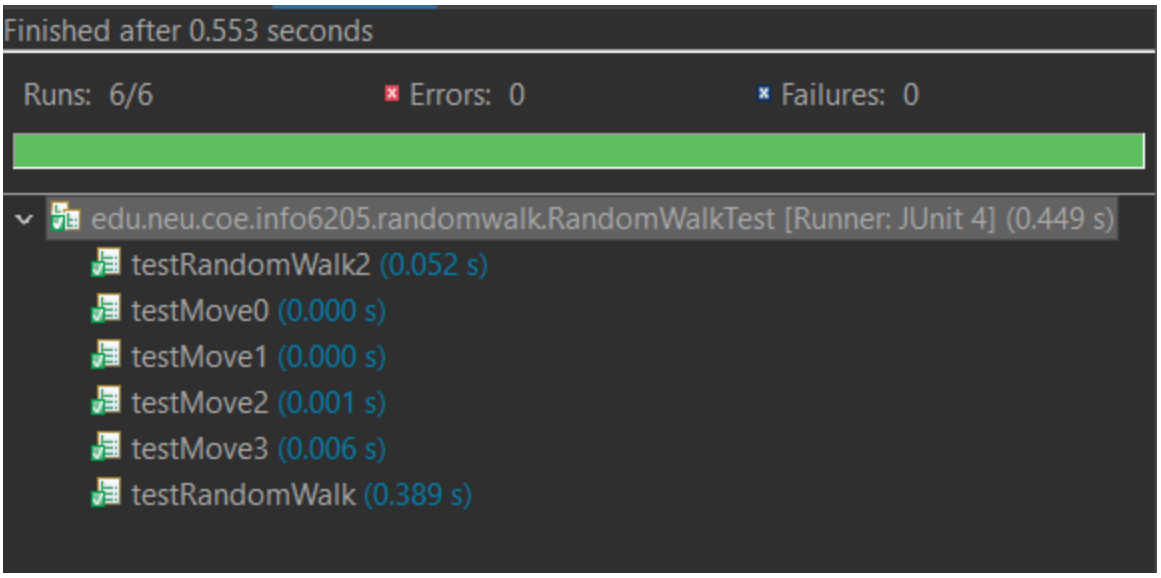
```
package edu.neu.coe.info6205.randomwalk;

import java.util.Random;


public class RandomWalk {


   private int x = 0;
```

```
private int y = 0;

private final Random random = new Random();

/**
 * Private method to move the current position, that's to say the drunkard moves
 *
 * @param dx the distance he moves in the x direction
 * @param dy the distance he moves in the y direction
 */
private void move(int dx, int dy) {
   this.x += dx;
   this.y += dy;
}

/**
 * Perform a random walk of m steps
 *
 * @param m the number of steps the drunkard takes
 */
private void randomWalk(int m) {
        for (int i = 0; i < m; i++) {
                this.randomMove();
        }
}

/**
```

```
    * Private method to generate a random move according to the rules of the situation.
    * That's to say, moves can be (+-1, 0) or (0, +-1).
    */
   private void randomMove() {
      boolean ns = random.nextBoolean();
      int step = random.nextBoolean() ? 1 : -1;
      move(ns ? step : 0, ns ? 0 : step);
   }

   /**
    * Method to compute the distance from the origin (the lamp-post where the drunkard
starts) to his current position.
    *
    * @return the (Euclidean) distance from the origin to the current position.
    */
   public double distance() {
      double xDiffSquared = Math.pow(this.x, 2);
      double yDiffSquared = Math.pow(this.y, 2);
      return Math.sqrt(xDiffSquared + yDiffSquared);
   }

   /**
    * Perform multiple random walk experiments, returning the mean distance.
    *
    * @param m the number of steps for each experiment
    * @param n the number of experiments to run
    * @return the mean distance
```

```
   */
   public static double randomWalkMulti(int m, int n) {
      double totalDistance = 0;
      for (int i = 0; i < n; i++) {
         RandomWalk walk = new RandomWalk();
         walk.randomWalk(m);
         totalDistance = totalDistance + walk.distance();
      }
      return totalDistance / n;
   }

   public static void main(String[] args) {
      if (args.length == 0)
         throw new RuntimeException("Syntax: RandomWalk steps [experiments]");
      int m = Integer.parseInt(args[0]);
      int n = 1000;
      if (args.length > 1) n = Integer.parseInt(args[1]);
      double meanDistance = randomWalkMulti(m, n);
      System.out.println(m + " steps: " + meanDistance + " over " + n + " experiments");
   }

}
```

Screenshot of unit tests passing:


Finished after 0.553 seconds
Runs: 6/6
Errors: 0
Failures: 0
edu.neu.coe.info6205.randomwalk.RandomWalkTest [Runner: JUnit 4] (0.449 s)
testRandomWalk2 (0.052 s)
testMove0 (0.000 s)
testMove1 (0.000 s)
testMove2 (0.001 s)
testMove3 (0.006 s)
testRandomWalk (0.389 s)