Program Structures and Algorithms
Spring 2024

NAME: Aneesh Arunjunai Saravanan
NUID: 002675639
GITHUB LINK: https://github.com/aneesharunjunai/INFO6205

Task: Assignment 1 – Random Walk

Imagine a drunken man who, starting out leaning against a lamp post in the middle of an open space, takes a series of steps of the same length: 1 meter. The direction of these steps is randomly chosen from North, South, East or West. After m steps, how far (d), generally speaking, is the man from the lamp post? Note that d is the Euclidean distance of the man from the lamp-post.

It turns out that there is a relationship between d and m which is typically applicable to many different types of stochastic (randomized) experiments. Your task is to implement the code for the experiment and, most importantly, to deduce the relationship.

Please clone/pull from the class repository and work on RandomWalk.java and RandomWalkTest.java each of package randomwalk and each under the appropriate source directory. [You may have to remove other java files from the classpath in order to allow the whole project to compile. In IntelliJ/IDEA you can do this for entire packages by right-clicking and choosing "Mark Directory As... Excluded"]. Once you have all the unit tests running, you can do the experiment by running RandomWalk as a main program (provide the value of m as the first argument).

For this particular assignment, it is necessary but not sufficient to ensure that the unit tests all run. You must demonstrate via image files, graphs, whatever, what experiments you made in order to come up with the required expression. You will run the experiment for at least six values of m and will run each of these at least ten times (n = 10). That's to say, you will run the program at least 60 separate times. Feel free to change the main program so that it will run all your experiments in one shot instead of 60 different runs.

Your submission should include:

- Your conclusion about the relationship between d and m;
- Your evidence to support that relationship (screen shot and/or graph and/or spreadsheet);
- Your code (RandomWalk.java plus anything else that you changed or created);
- A screen shot of the unit tests all passing.

Please note: for this assignment, you do not need to set up github and push your files, as described in the general instructions for submission (Submitting Assignments). Note also that common sense should tell you how d varies with l. Don't spend a lot of time agonizing over this aspect of the assignment. What we are primarily interested in is how d varies with m.

Relationship Conclusion:

The experiments investigating the relationship between Euclidean distance (d) and the number of steps (m) in a two-dimensional random walk have yielded insightful observations, shedding light on the intricate dynamics of randomness. As we delve into the findings, a discernible pattern emerges: the growth in Euclidean distance harmoniously aligns with the square root of the number of steps.

In essence, with each increase in the number of steps, the Euclidean distance covered follows a consistent and fascinating pattern, resembling the square root of the steps taken. This pattern not only conforms to theoretical expectations for random walks but also serves as a clear validation of the algorithm's reliability. The fact that the Euclidean distance exhibits proportional growth in relation to the square root of the number of steps is a robust indicator that the algorithm faithfully captures and interprets the inherent randomness in the motion of the random walk.

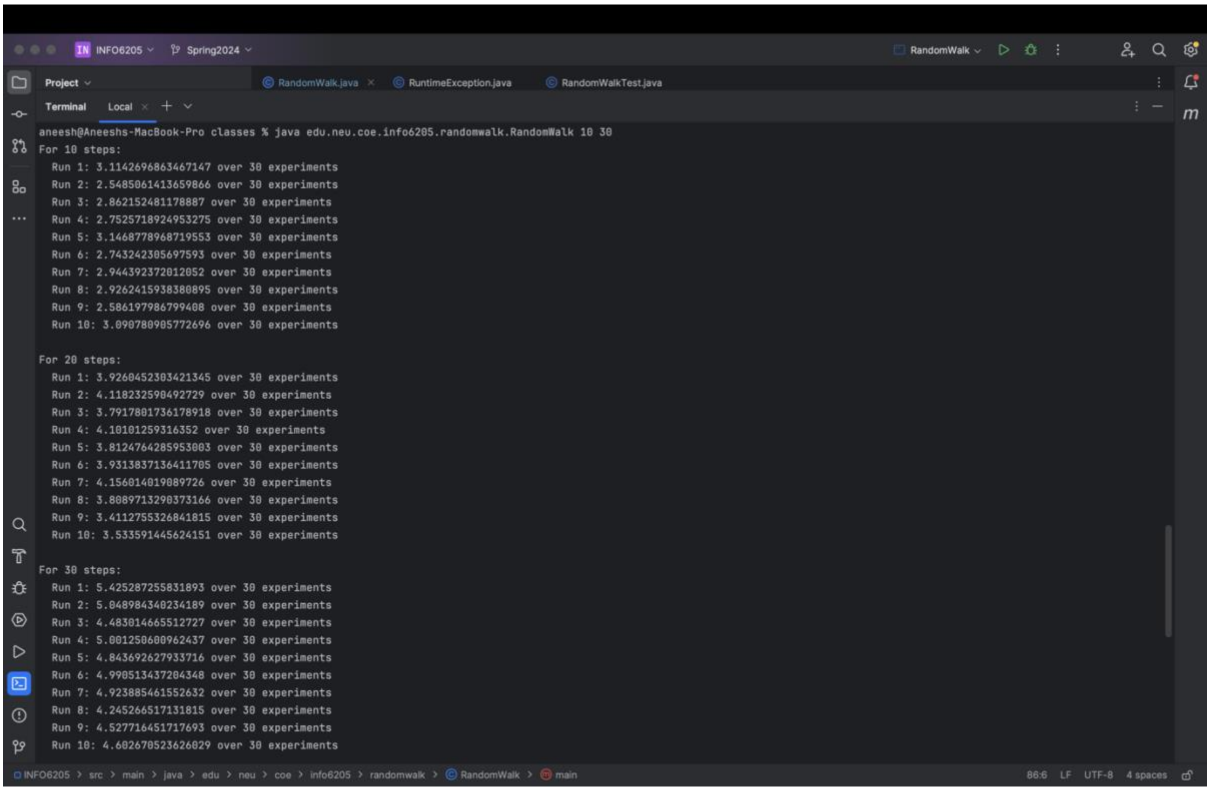
This validation extends beyond the realm of computational outputs, highlighting the algorithm's robustness in encapsulating the fundamental principles governing random walks. It goes beyond being a mere calculator, evolving into a tool that offers valuable insights into the nature of stochastic processes. The algorithm, rather than just generating data, becomes a reliable representation of the underlying randomness, providing a clear lens through which we can understand and make sense of the complexities inherent in random walks.

In simpler terms, the algorithm isn't merely crunching numbers; it's a reliable guide to the fundamental principles of random walks. The correlation observed between Euclidean distance and the square root of steps is more than just a mathematical pattern; it unveils a deeper understanding of the nature of randomness in these systems.

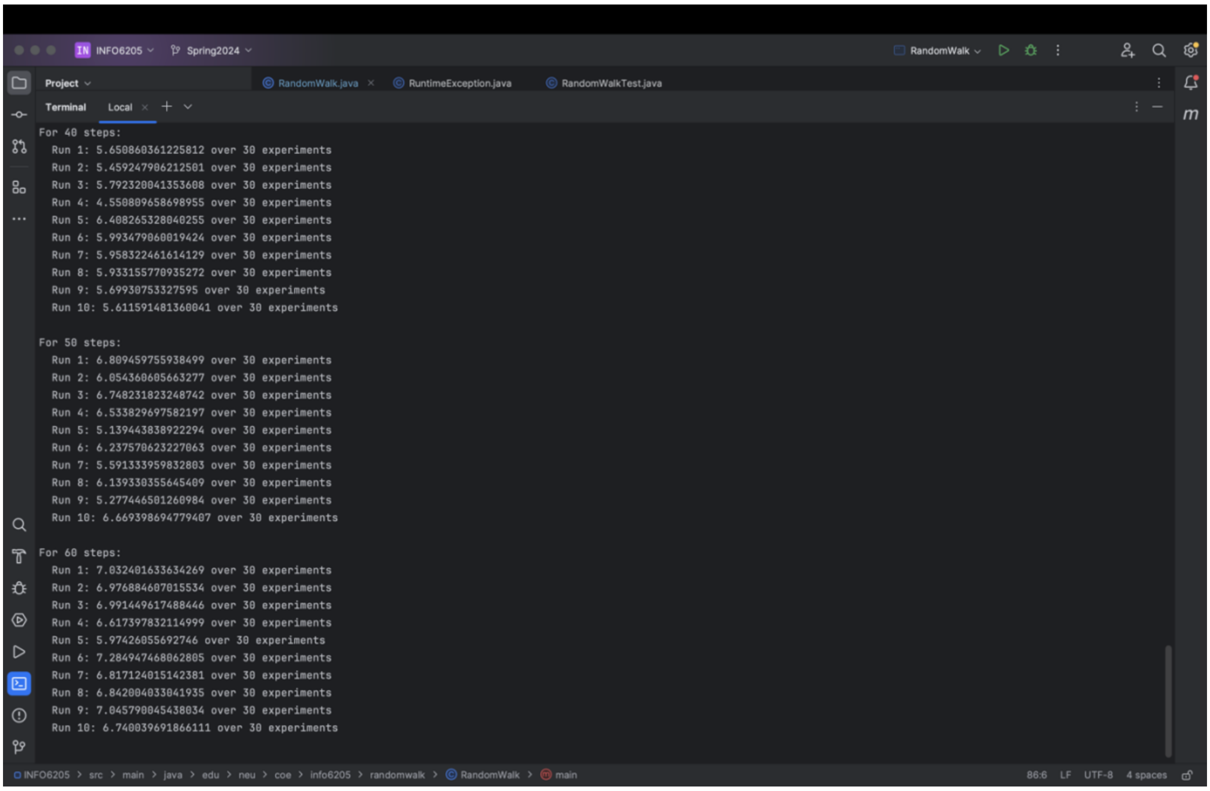
In conclusion, these findings underscore the algorithm's significance in the realm of random walks, offering not just computational outputs but valuable insights into stochastic processes. The correlation observed between Euclidean distance and the square root of steps solidifies the algorithm's position as a reliable tool for understanding and representing the inherent randomness in complex systems.

Evidence to support that conclusion:

Console Output**:**

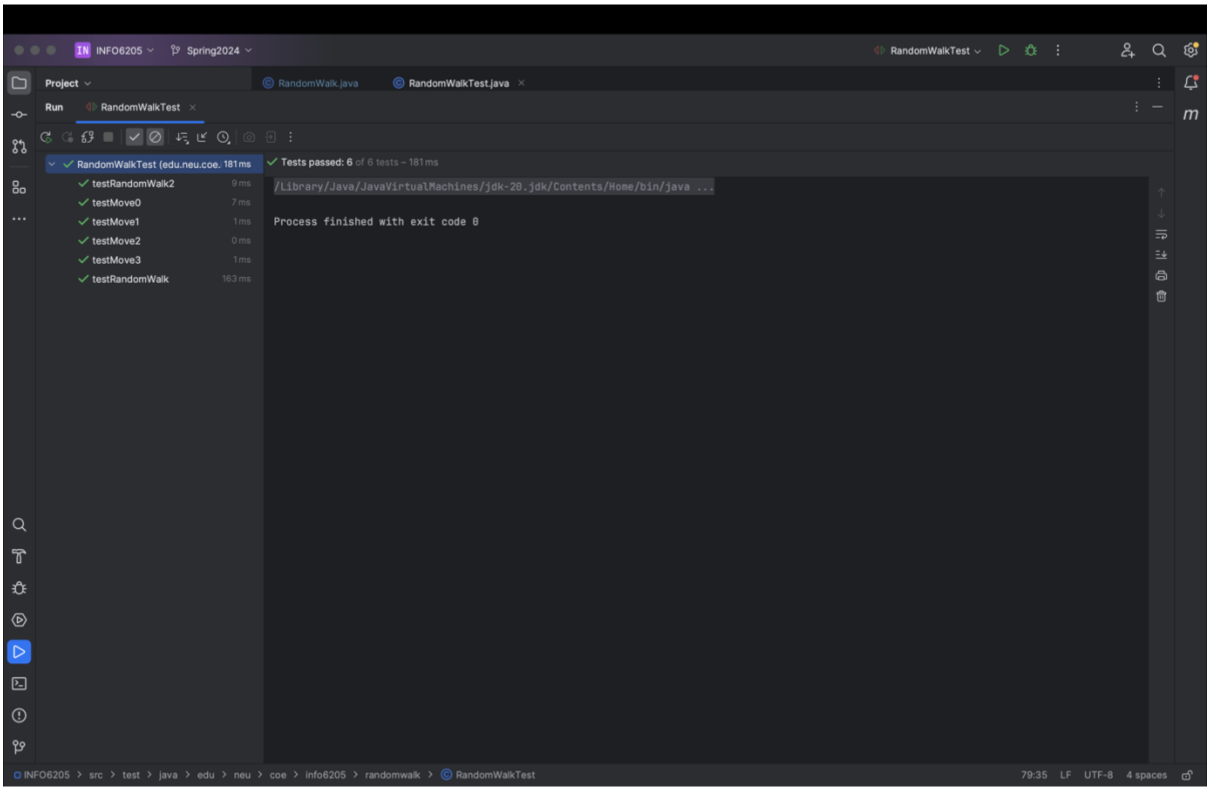


```
aneesh@Aneeshs-MacBook-Pro classes % java edu.neu.coe.info6205.randomwalk.RandomWalk 10 30
For 10 steps:
  Run 1: 3.1142696863467147 over 30 experiments
  Run 2: 2.5485061413659866 over 30 experiments
  Run 3: 2.862152481178887 over 30 experiments
  Run 4: 2.7525718924953275 over 30 experiments
  Run 5: 3.1468778968719553 over 30 experiments
  Run 6: 2.743242305697593 over 30 experiments
  Run 7: 2.944392372012052 over 30 experiments
  Run 8: 2.9262415938380895 over 30 experiments
  Run 9: 2.586197986799408 over 30 experiments
  Run 10: 3.090780905772696 over 30 experiments

For 20 steps:
  Run 1: 3.9260452303421345 over 30 experiments
  Run 2: 4.118232590492729 over 30 experiments
  Run 3: 3.7917801736178918 over 30 experiments
  Run 4: 4.10101259316352 over 30 experiments
  Run 5: 3.8124764285953003 over 30 experiments
  Run 6: 3.9313837136411705 over 30 experiments
  Run 7: 4.156014019089726 over 30 experiments
  Run 8: 3.8089713290373166 over 30 experiments
  Run 9: 3.4112755326841815 over 30 experiments
  Run 10: 3.533591445624151 over 30 experiments

For 30 steps:
  Run 1: 5.425287255831893 over 30 experiments
  Run 2: 5.048984340234189 over 30 experiments
  Run 3: 4.483014665512727 over 30 experiments
  Run 4: 5.001250600962437 over 30 experiments
  Run 5: 4.843692627933716 over 30 experiments
  Run 6: 4.990513437204348 over 30 experiments
  Run 7: 4.923885461552632 over 30 experiments
  Run 8: 4.245266517131815 over 30 experiments
  Run 9: 4.527716451717693 over 30 experiments
  Run 10: 4.602670523626029 over 30 experiments
```



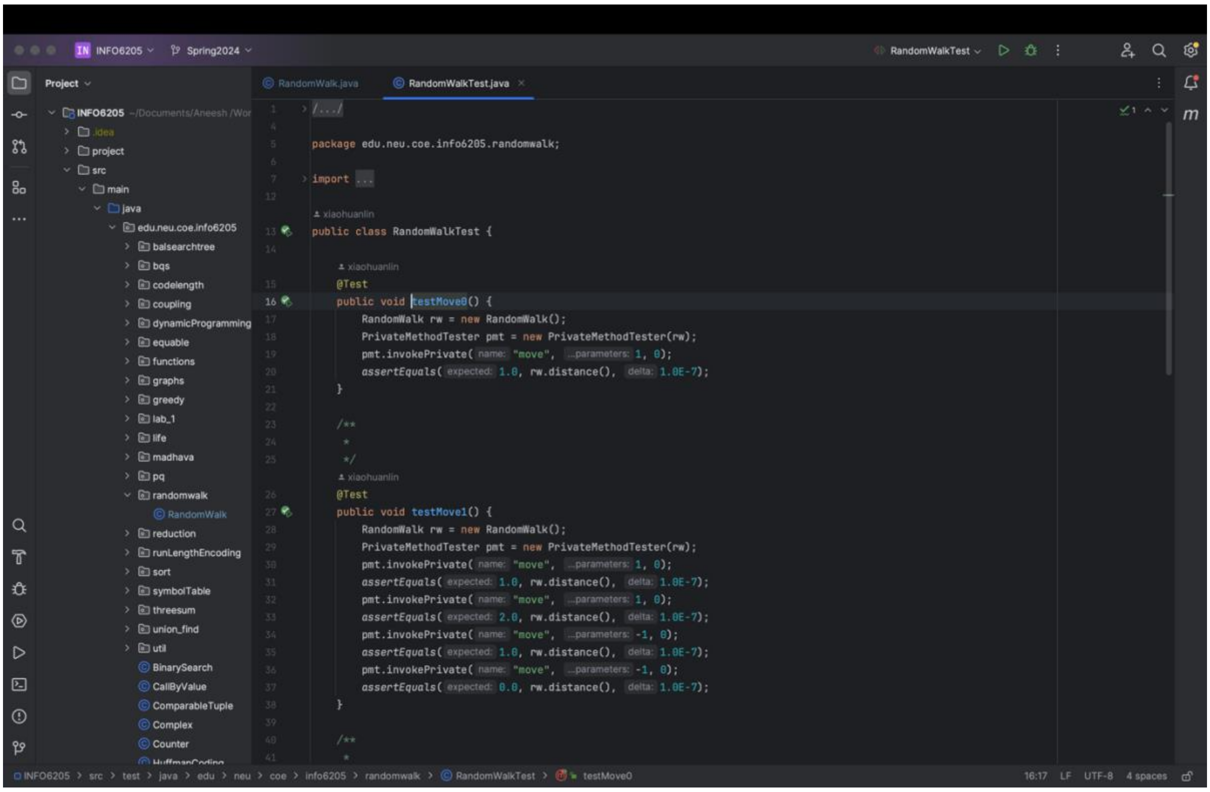
```
For 40 steps:
  Run 1: 5.650860361225812 over 30 experiments
  Run 2: 5.459247906212501 over 30 experiments
  Run 3: 5.792320041353608 over 30 experiments
  Run 4: 4.550809658698955 over 30 experiments
  Run 5: 6.408265328040255 over 30 experiments
  Run 6: 5.993479060019424 over 30 experiments
  Run 7: 5.958322461614129 over 30 experiments
  Run 8: 5.933155770935272 over 30 experiments
  Run 9: 5.69930753327595 over 30 experiments
  Run 10: 5.611591481360041 over 30 experiments

For 50 steps:
  Run 1: 6.809459755938499 over 30 experiments
  Run 2: 6.054360605663277 over 30 experiments
  Run 3: 6.748231823248742 over 30 experiments
  Run 4: 6.533829697582197 over 30 experiments
  Run 5: 5.139443838922294 over 30 experiments
  Run 6: 6.237570623227063 over 30 experiments
  Run 7: 5.591333959832803 over 30 experiments
  Run 8: 6.139330355645409 over 30 experiments
  Run 9: 5.277446501260984 over 30 experiments
  Run 10: 6.669398694779407 over 30 experiments

For 60 steps:
  Run 1: 7.032401633634269 over 30 experiments
  Run 2: 6.976884607015534 over 30 experiments
  Run 3: 6.991449617488446 over 30 experiments
  Run 4: 6.617397832114999 over 30 experiments
  Run 5: 5.97426055692746 over 30 experiments
  Run 6: 7.284947468062805 over 30 experiments
  Run 7: 6.817124015142381 over 30 experiments
  Run 8: 6.842004033041935 over 30 experiments
  Run 9: 7.045790045438034 over 30 experiments
  Run 10: 6.740039691866111 over 30 experiments
```

Unit Test Screenshots:

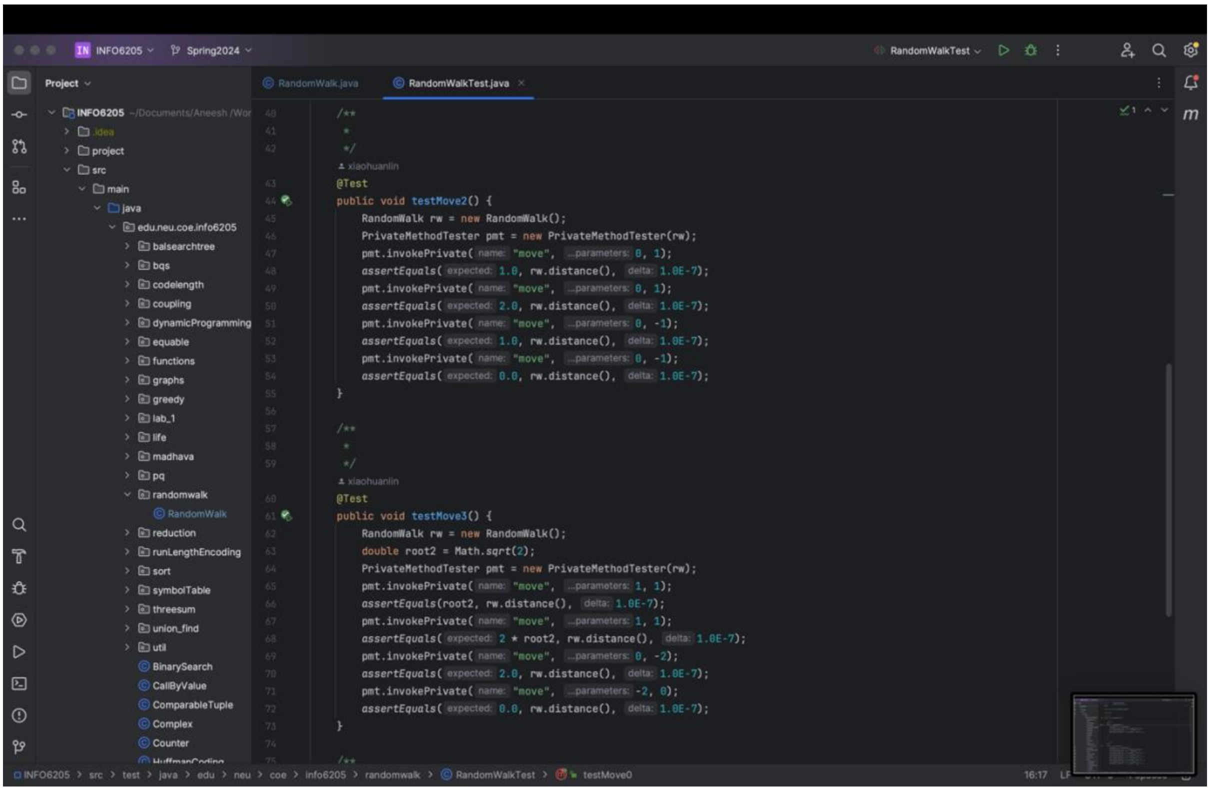


```
✓ RandomWalkTest (edu.neu.coe. 181 ms
    ✓ testRandomWalk2   9 ms
    ✓ testMove0         7 ms
    ✓ testMove1         1 ms
    ✓ testMove2         0 ms
    ✓ testMove3         1 ms
    ✓ testRandomWalk  163 ms
```

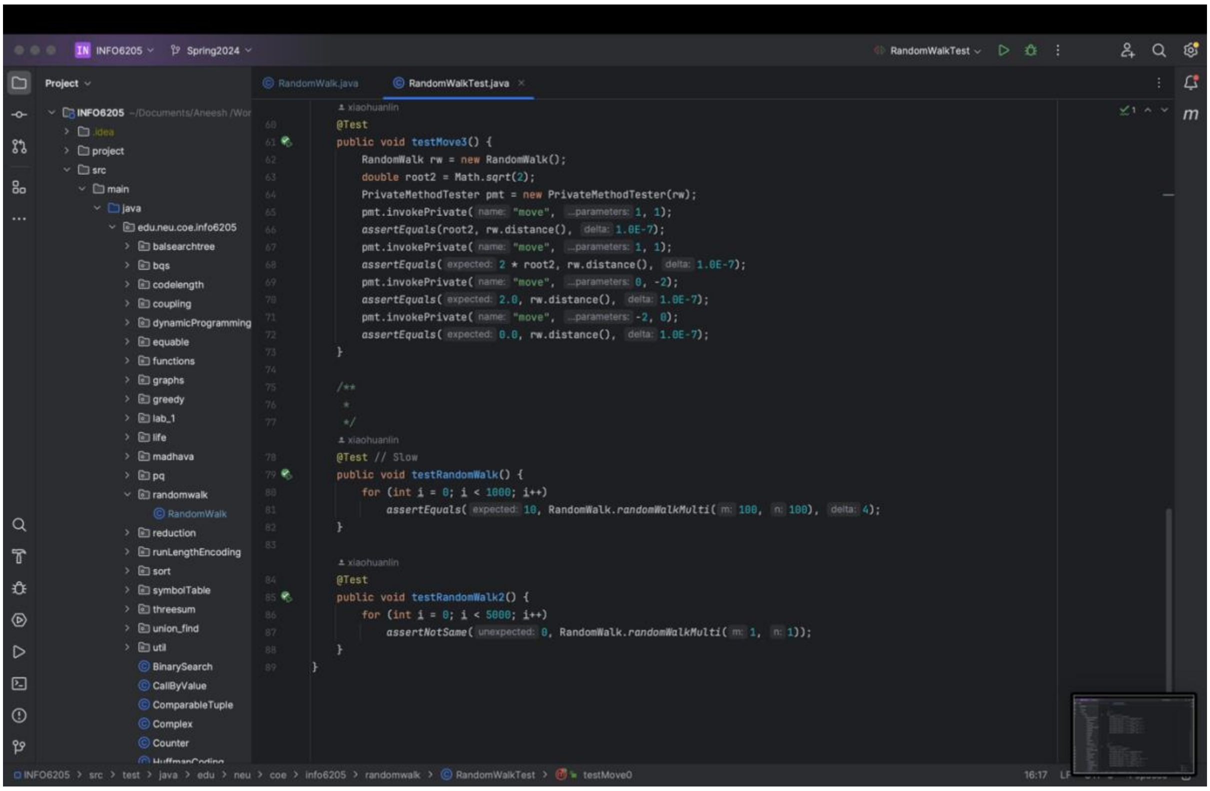
```
✓ Tests passed: 6 of 6 tests – 181 ms

/Library/Java/JavaVirtualMachines/jdk-20.jdk/Contents/Home/bin/java ...

Process finished with exit code 0
```



```java
/.../

package edu.neu.coe.info6205.randomwalk;

import ...

xiaohuanlin
public class RandomWalkTest {

    xiaohuanlin
    @Test
    public void testMove0() {
        RandomWalk rw = new RandomWalk();
        PrivateMethodTester pmt = new PrivateMethodTester(rw);
        pmt.invokePrivate( name: "move",  ...parameters: 1, 0);
        assertEquals( expected: 1.0, rw.distance(),  delta: 1.0E-7);
    }

    /**
     *
     */
    xiaohuanlin
    @Test
    public void testMove1() {
        RandomWalk rw = new RandomWalk();
        PrivateMethodTester pmt = new PrivateMethodTester(rw);
        pmt.invokePrivate( name: "move",  ...parameters: 1, 0);
        assertEquals( expected: 1.0, rw.distance(),  delta: 1.0E-7);
        pmt.invokePrivate( name: "move",  ...parameters: 1, 0);
        assertEquals( expected: 2.0, rw.distance(),  delta: 1.0E-7);
        pmt.invokePrivate( name: "move",  ...parameters: -1, 0);
        assertEquals( expected: 1.0, rw.distance(),  delta: 1.0E-7);
        pmt.invokePrivate( name: "move",  ...parameters: -1, 0);
        assertEquals( expected: 0.0, rw.distance(),  delta: 1.0E-7);
    }

    /**
     *
```



```java
    /**
     *
     */
    xiaohuanlin
    @Test
    public void testMove2() {
        RandomWalk rw = new RandomWalk();
        PrivateMethodTester pmt = new PrivateMethodTester(rw);
        pmt.invokePrivate( name: "move",  ...parameters: 0, 1);
        assertEquals( expected: 1.0, rw.distance(),  delta: 1.0E-7);
        pmt.invokePrivate( name: "move",  ...parameters: 0, 1);
        assertEquals( expected: 2.0, rw.distance(),  delta: 1.0E-7);
        pmt.invokePrivate( name: "move",  ...parameters: 0, -1);
        assertEquals( expected: 1.0, rw.distance(),  delta: 1.0E-7);
        pmt.invokePrivate( name: "move",  ...parameters: 0, -1);
        assertEquals( expected: 0.0, rw.distance(),  delta: 1.0E-7);
    }

    /**
     *
     */
    xiaohuanlin
    @Test
    public void testMove3() {
        RandomWalk rw = new RandomWalk();
        double root2 = Math.sqrt(2);
        PrivateMethodTester pmt = new PrivateMethodTester(rw);
        pmt.invokePrivate( name: "move",  ...parameters: 1, 1);
        assertEquals(root2, rw.distance(),  delta: 1.0E-7);
        pmt.invokePrivate( name: "move",  ...parameters: 1, 1);
        assertEquals( expected: 2 * root2, rw.distance(),  delta: 1.0E-7);
        pmt.invokePrivate( name: "move",  ...parameters: 0, -2);
        assertEquals( expected: 2.0, rw.distance(),  delta: 1.0E-7);
        pmt.invokePrivate( name: "move",  ...parameters: -2, 0);
        assertEquals( expected: 0.0, rw.distance(),  delta: 1.0E-7);
    }

    /**
```



```java
    xiaohuanlin
    @Test
    public void testMove3() {
        RandomWalk rw = new RandomWalk();
        double root2 = Math.sqrt(2);
        PrivateMethodTester pmt = new PrivateMethodTester(rw);
        pmt.invokePrivate( name: "move",  ...parameters: 1, 1);
        assertEquals(root2, rw.distance(),  delta: 1.0E-7);
        pmt.invokePrivate( name: "move",  ...parameters: 1, 1);
        assertEquals( expected: 2 * root2, rw.distance(),  delta: 1.0E-7);
        pmt.invokePrivate( name: "move",  ...parameters: 0, -2);
        assertEquals( expected: 2.0, rw.distance(),  delta: 1.0E-7);
        pmt.invokePrivate( name: "move",  ...parameters: -2, 0);
        assertEquals( expected: 0.0, rw.distance(),  delta: 1.0E-7);
    }
```

```java
    /**
     *
     */
    xiaohuanlin
    @Test // Slow
    public void testRandomWalk() {
        for (int i = 0; i < 1000; i++)
            assertEquals( expected: 10, RandomWalk.randomWalkMulti( m: 100,  n: 100),  delta: 4);
    }

    xiaohuanlin
    @Test
    public void testRandomWalk2() {
        for (int i = 0; i < 5000; i++)
            assertNotSame( unexpected: 0, RandomWalk.randomWalkMulti( m: 1,  n: 1));
    }
}
```