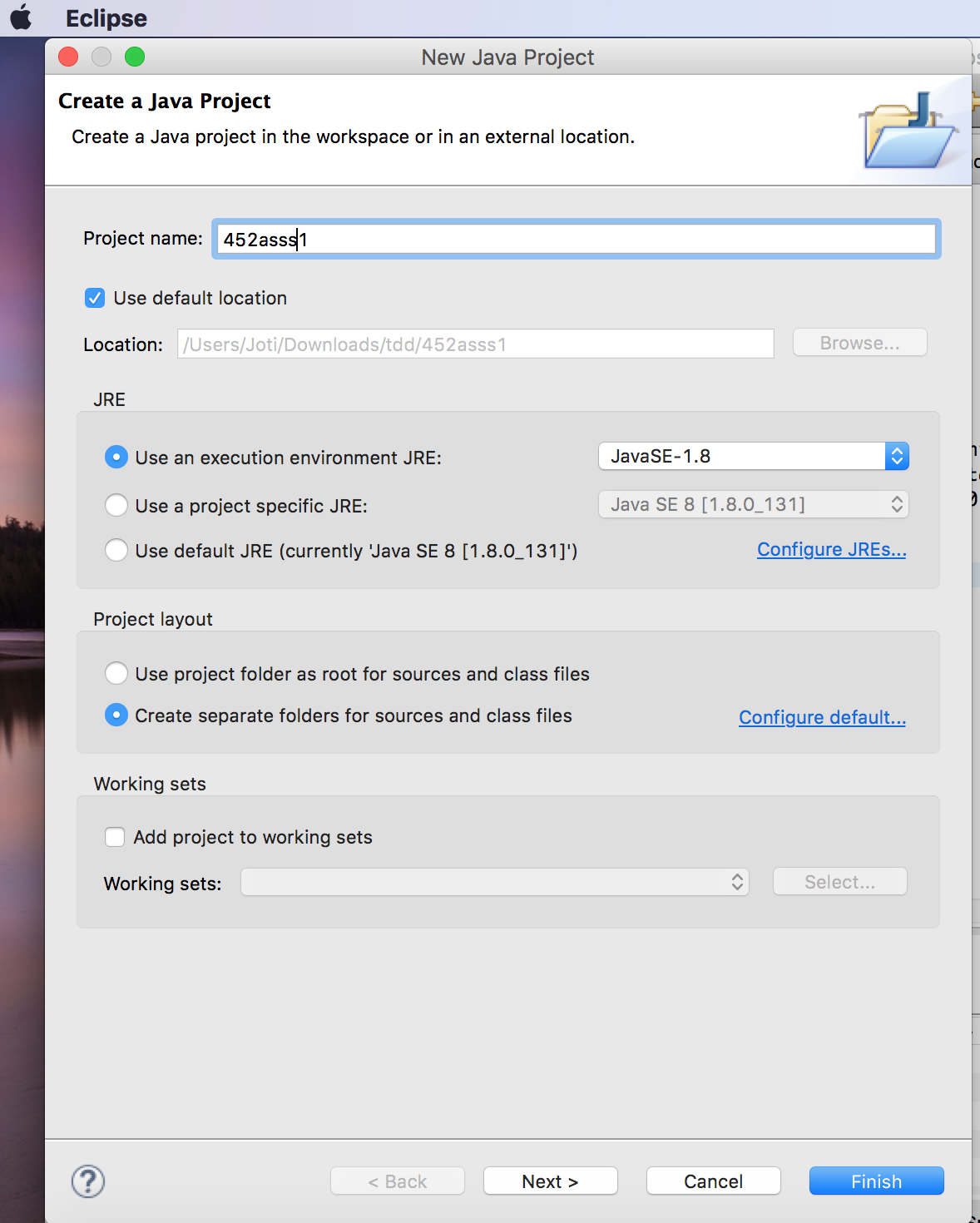
Git repository: <https://github.com/NB106/PRT452-Assignment-1>

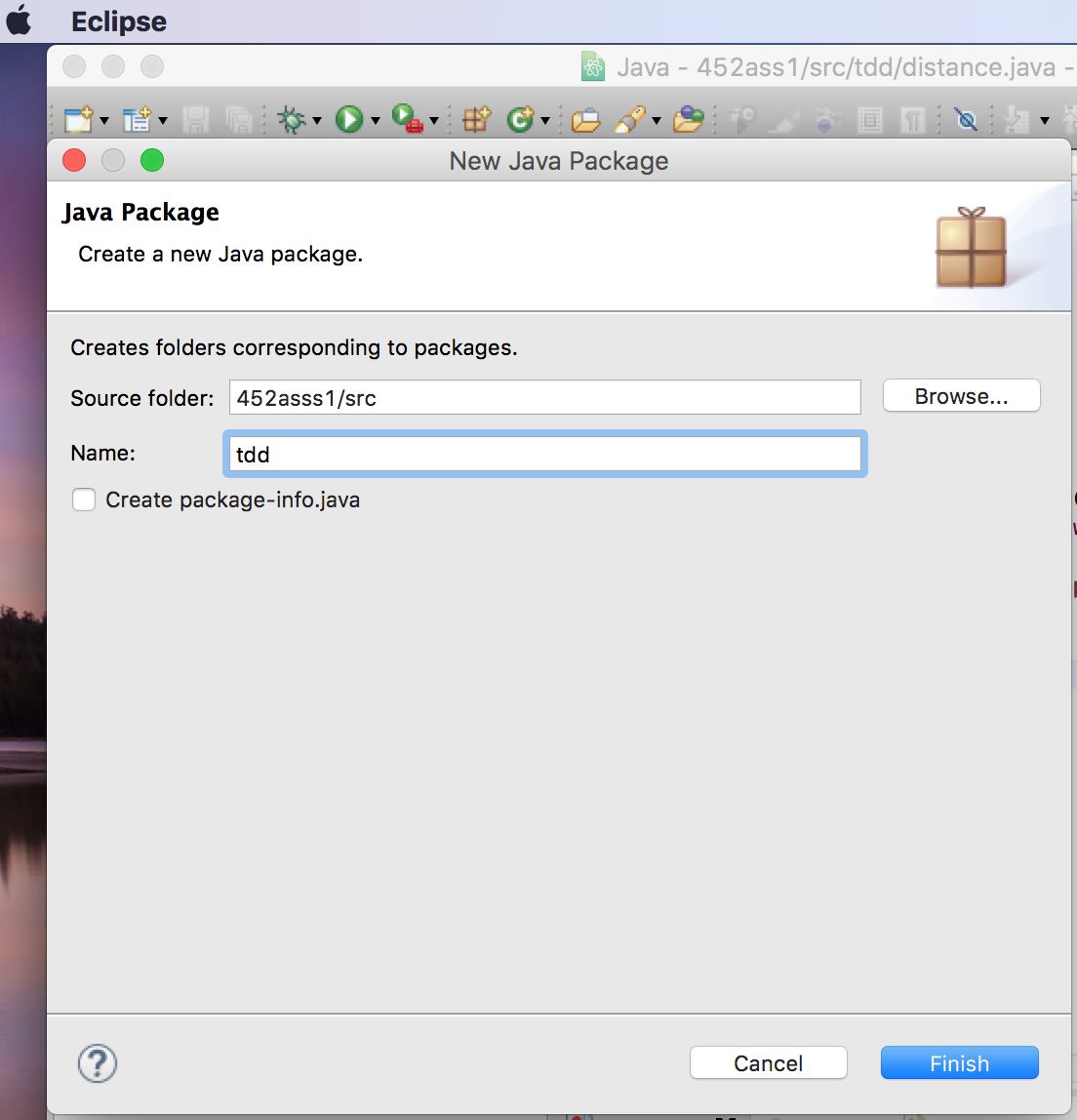
**Question 1: Unit test and Refactoring**

**Unit test**

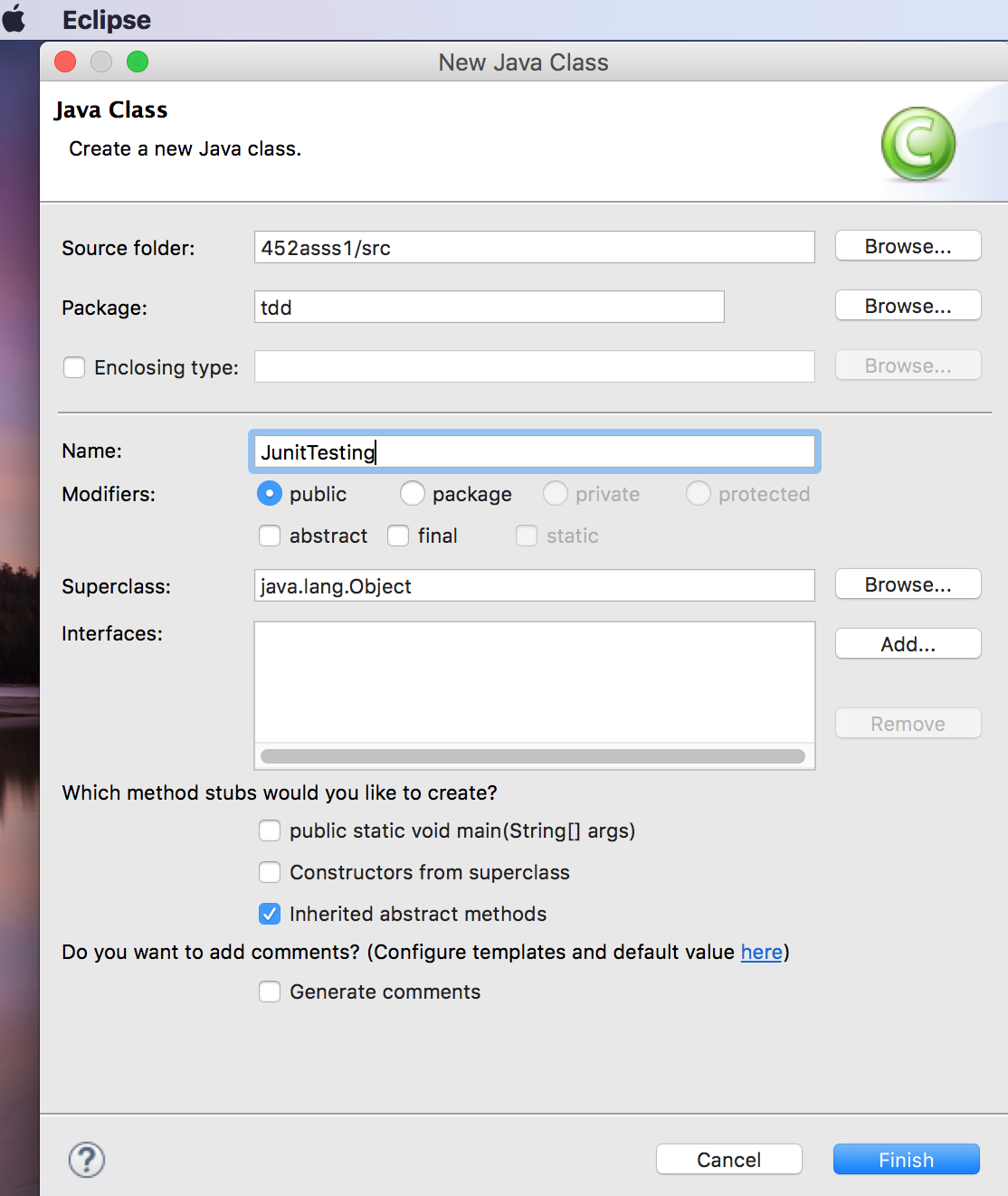
Below steps I took to create the program and test the program. For testing Junit method was used.

**Step 1: Creating a Java project on Eclipse**

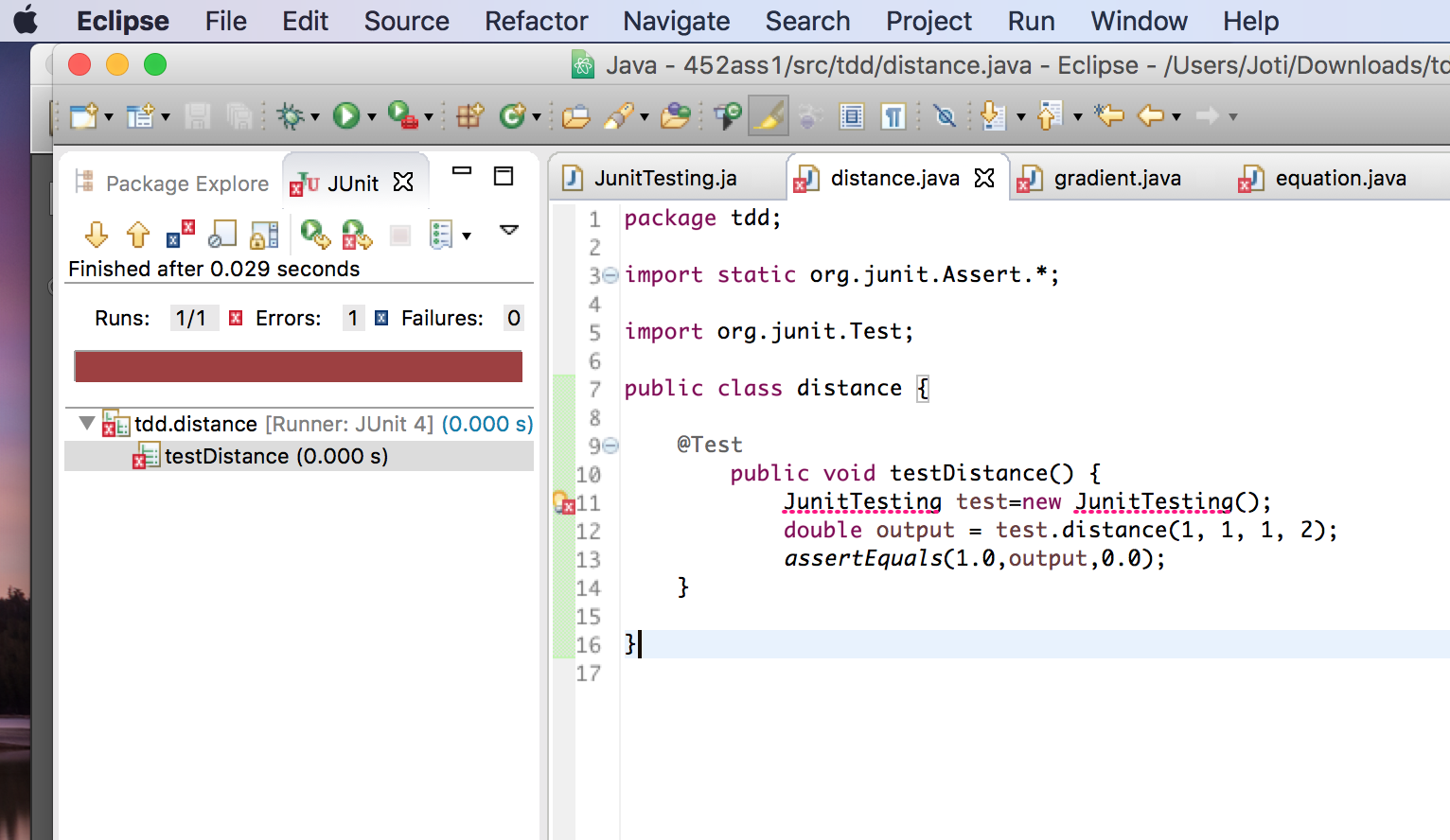


**Step 2: In the project I created a package named “tdd”**

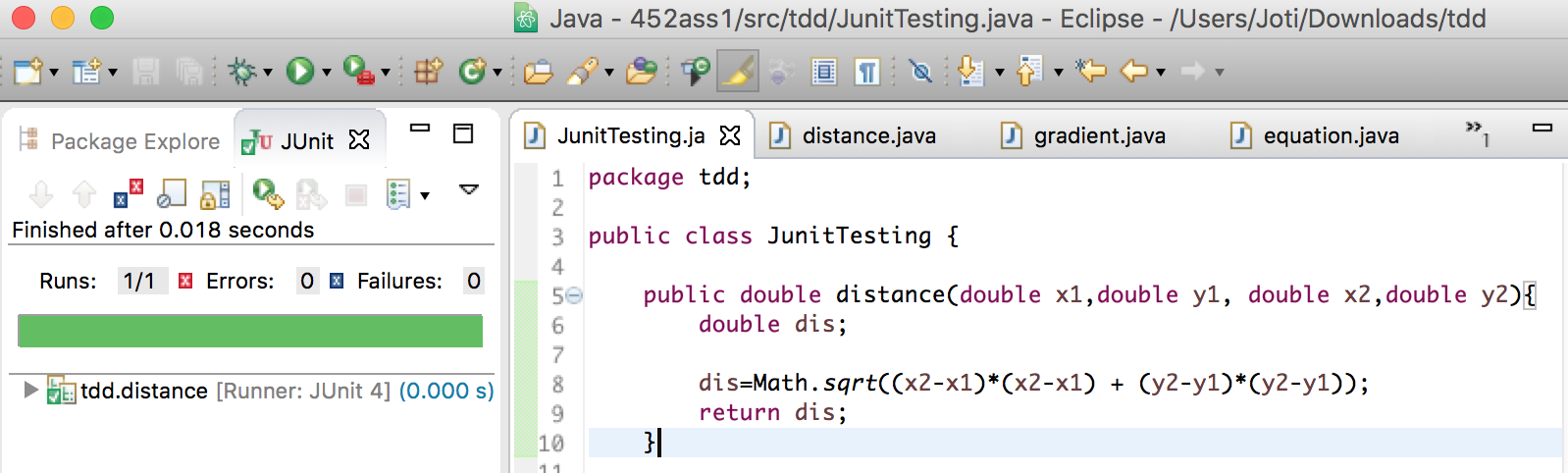
**Step 3: In the package tdd I created a class named JunitTesting**



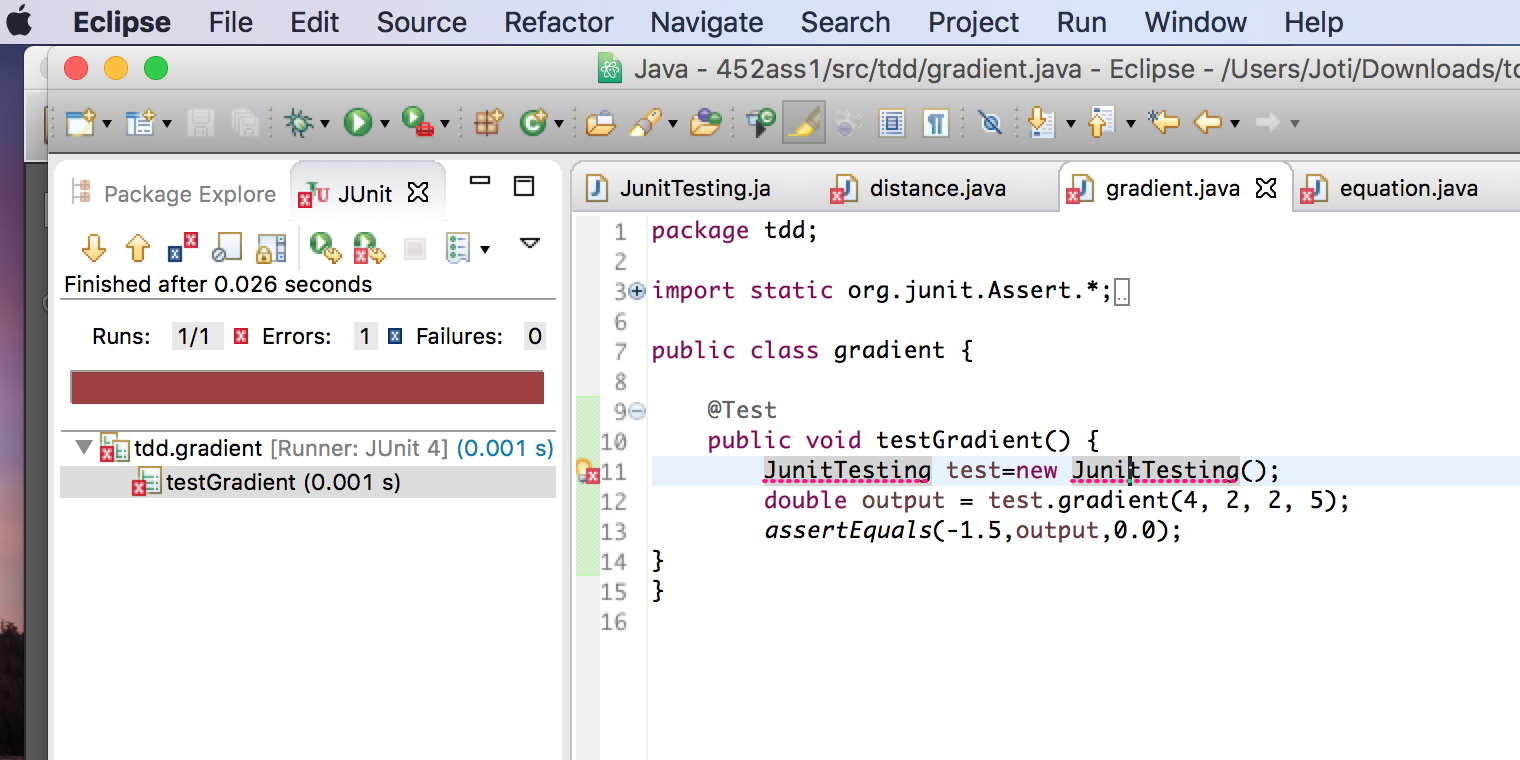
**Step 4: Created test for distance of the line and it shows error after running the program, as I didn’t write the real program.**



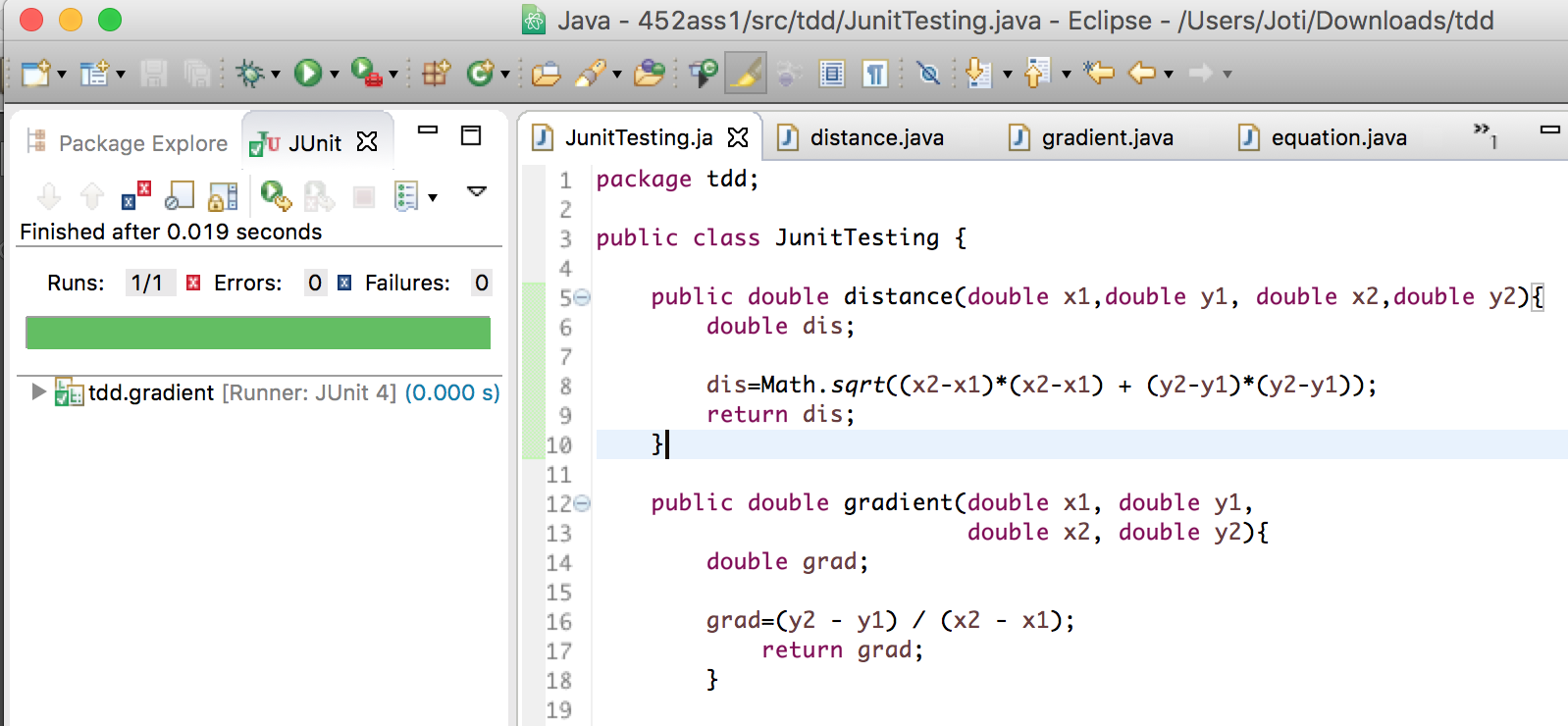
**Step 5: Then I wrote the code to find out the distance of the line and run the test again and did not show any error.**

****

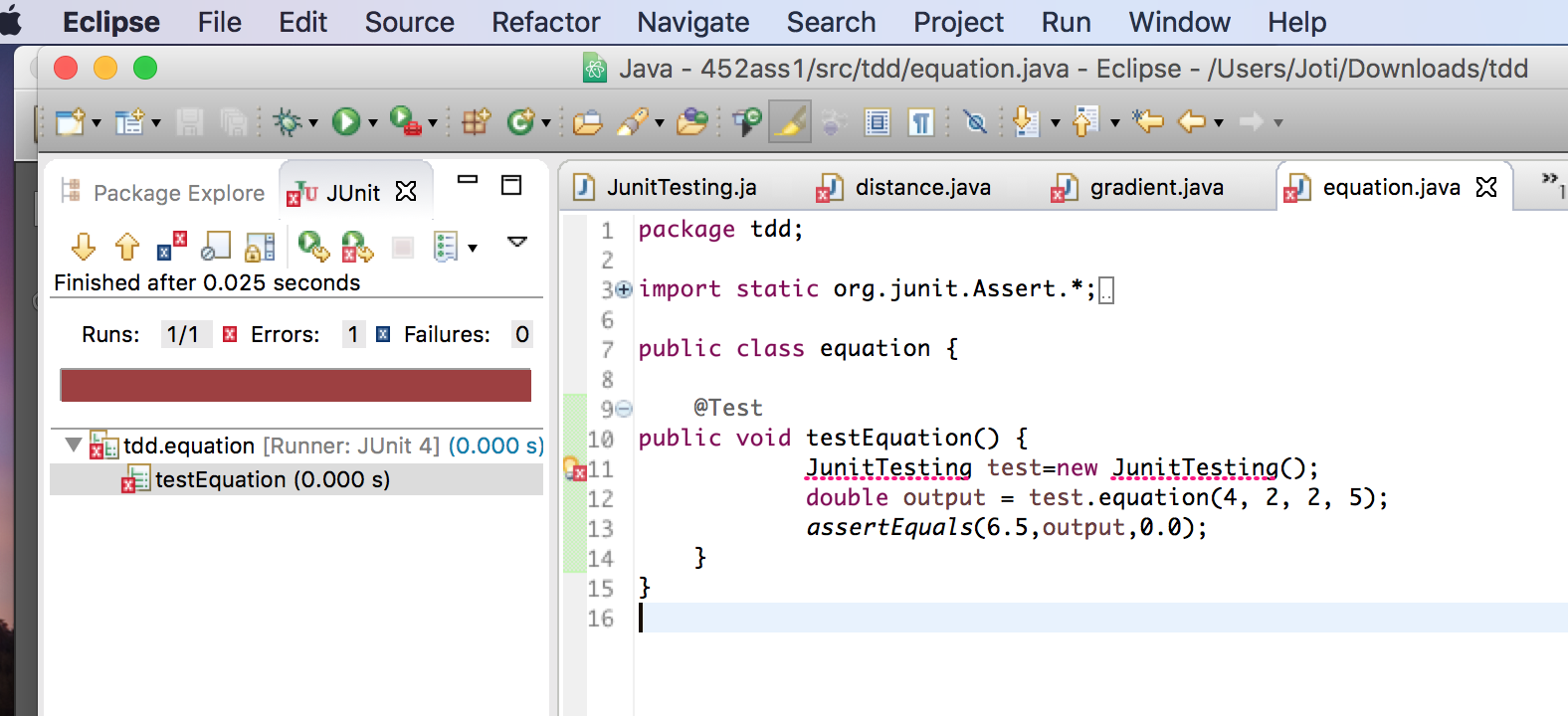
**Step 6: In this step I created test for gradient.**

****

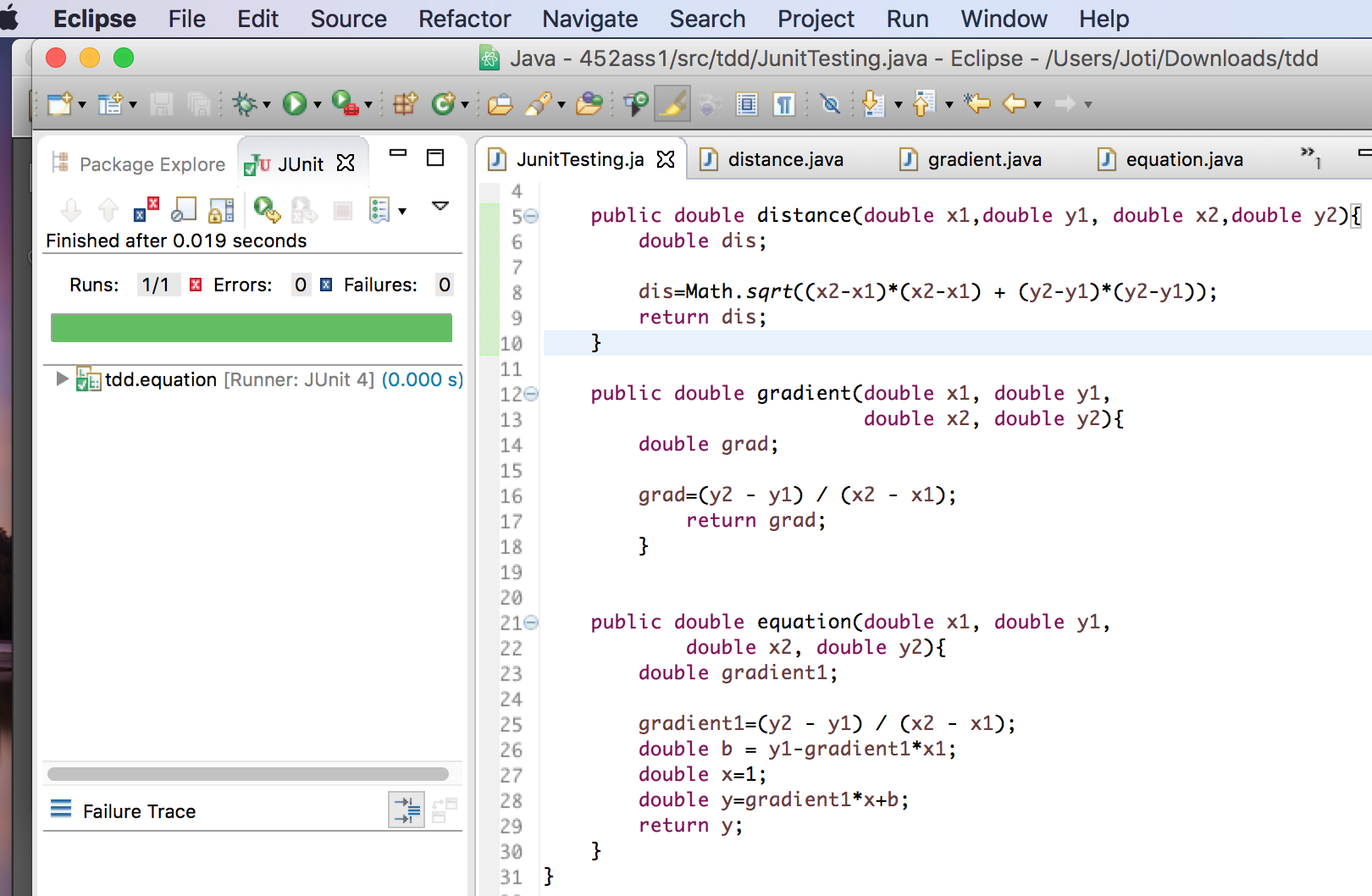
**Step 7: I wrote program to find out gradient. This time test did not show any error.**

****

**Step 8: I created test for Equation of the line, and showed error.**



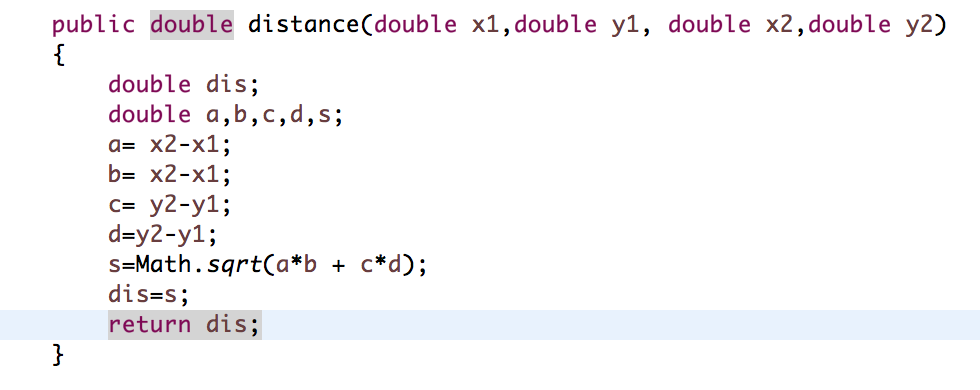
**Step 9: I wrote program to find out equation of the line and run the test again. This time no error was found.**



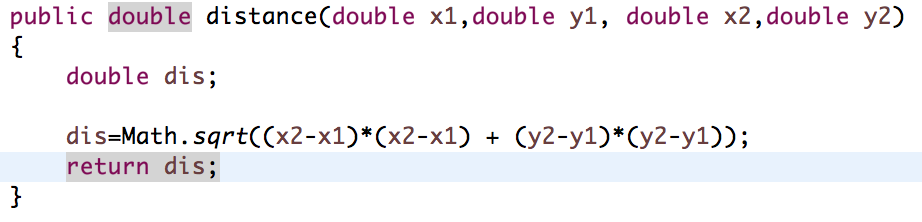
**Refactoring**

**Distance of the line:**

Before:

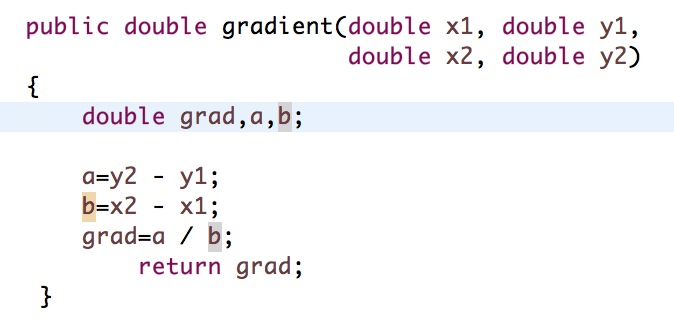
****

After:

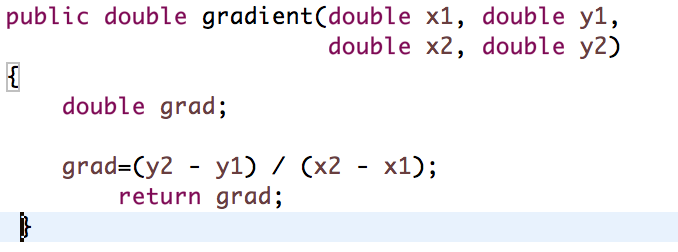
****

**Gradient of the line:**

Before refactoring:

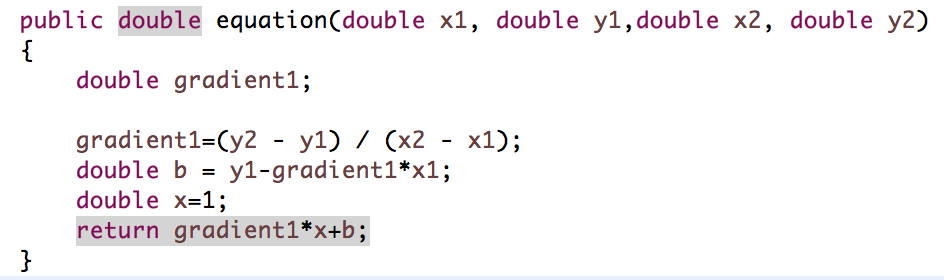


After refactoring:

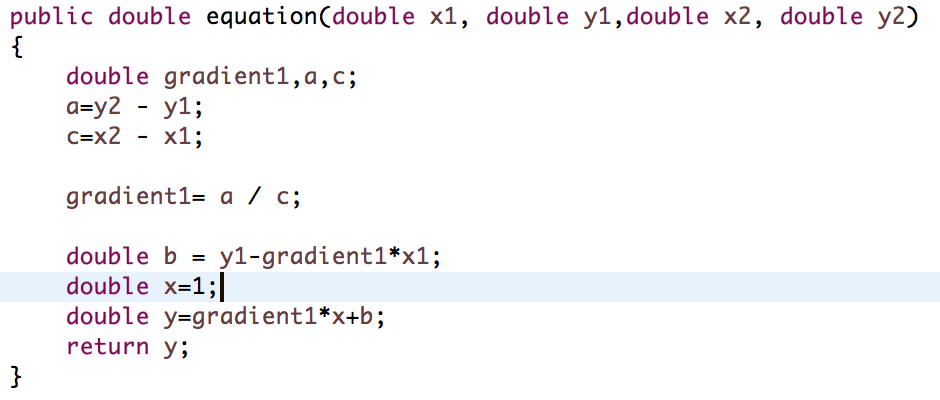


**Equation of the line:**

Before refactoring:



After refactoring:



**Question 2: Code smell**

5 issues and solutions:

**Issues:**

1. **Long methods:** Long methods make code hard to maintain and debug.
2. **Data Clumps:** Data clumps occur when the same group of data items (fields in classes, parameters in methods) reoccur in several places in a program.
3. **Duplicate Code:** Some same codes might be used several times in one program this can create bugs. When developers are not aware of the duplication, they only know to fix the occurrence they have come across.
4. **Switch:** Switch statements These often involve duplication, where the switch depends on the type of some value. The switch statements may be scattered around a program. In object-oriented languages, you can often use polymorphism to achieve the same thing.
5. When a class exists just to delegate to another, a developer should ask themselves what its real purpose is. Sometimes this is the result of a refactoring task, where logic has been moved out of a class gradually, leaving an almost empty shell.

**Solutions:**

1. **Long methods:** Breaking long methods into several smaller methods.
2. **Data Clumps:** To keep the group of parameters together, it can be useful to combine them together in a class. This can help aid organisation of code.
3. **Duplicate code:** The solution for this, can be implementing a function or method.
4. **Switch:** Using polymorphism cab a solution in object oriented programming to ignore this issue.
5. **Middle Man:** Solution is, for every class that exists, there is an overhead of maintenance. Make sure the class justifies its existence, and if it doesn't, go ahead and remove it.