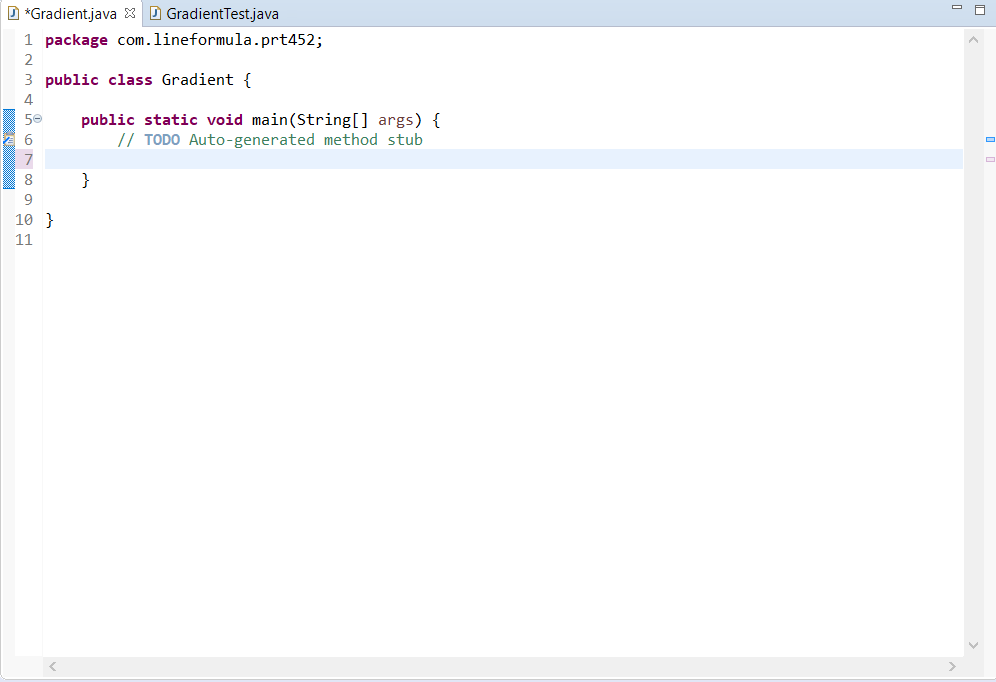
**PRT452 Assignment 1**

**Part 1**

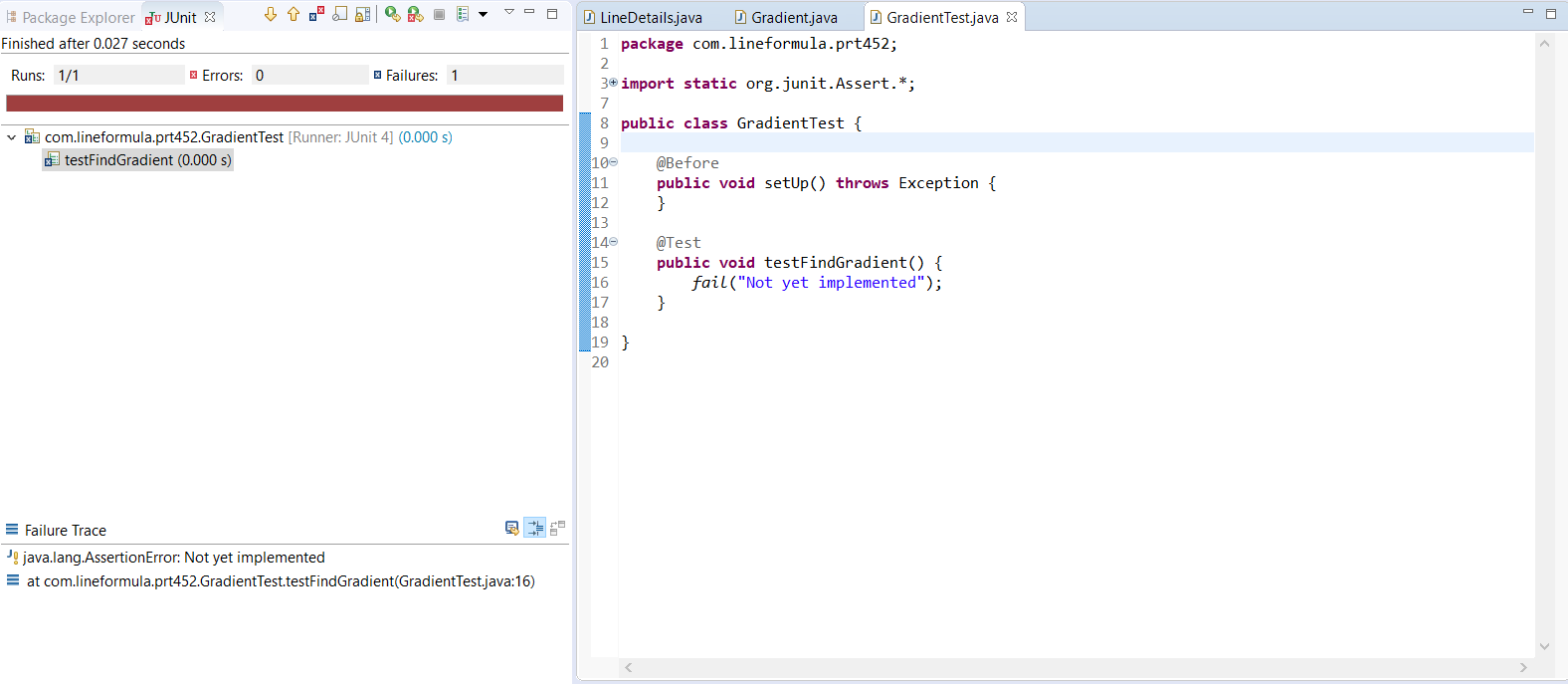
The goal of this part is to create a program that would return the gradient, distance, and equation of a line when two coordinates are taken as input. This program would be created using Test Driven Development, where tests are written before the code is written. My approach was to create and test each functionality in a separate class.

**Github:** [**https://github.com/PatrickA7/PRT452Assignment1**](https://github.com/PatrickA7/PRT452Assignment1)

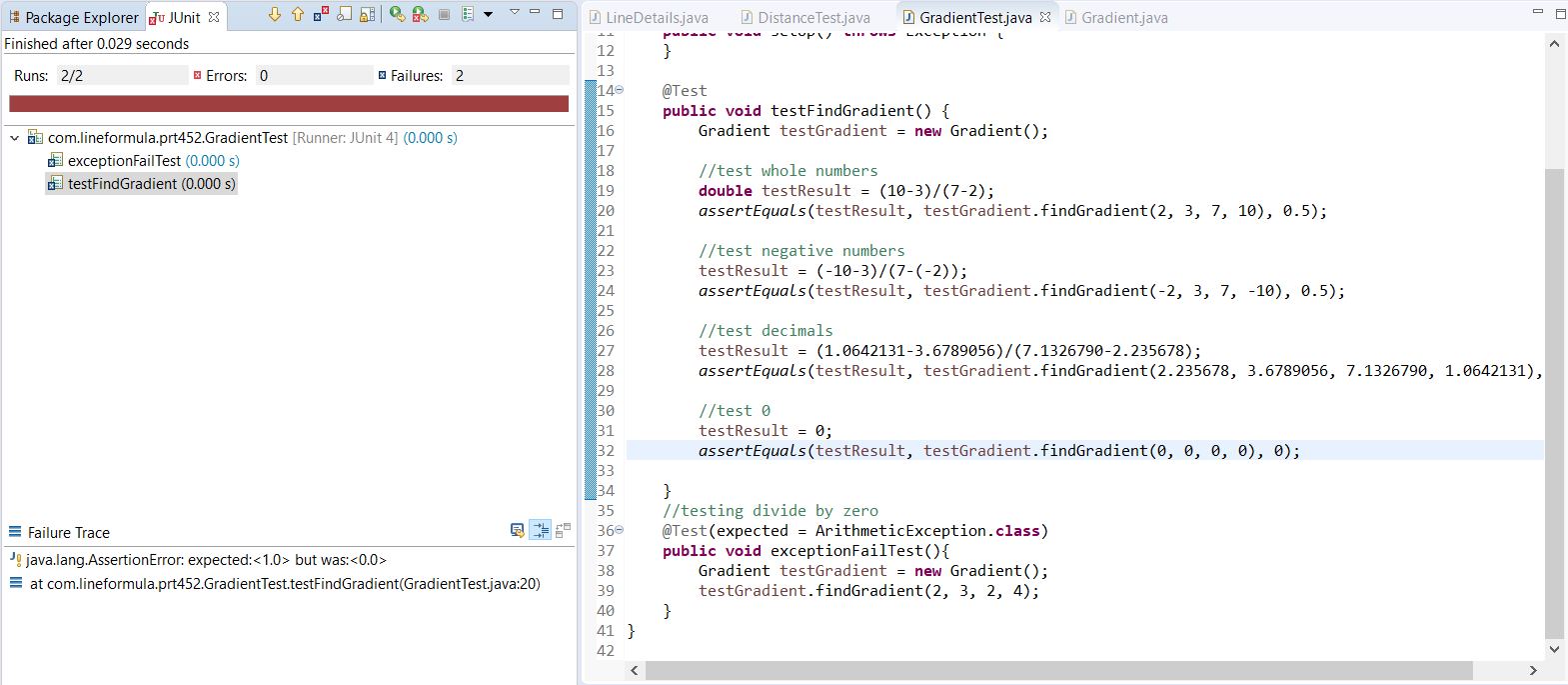
**Initial gradient class before code**



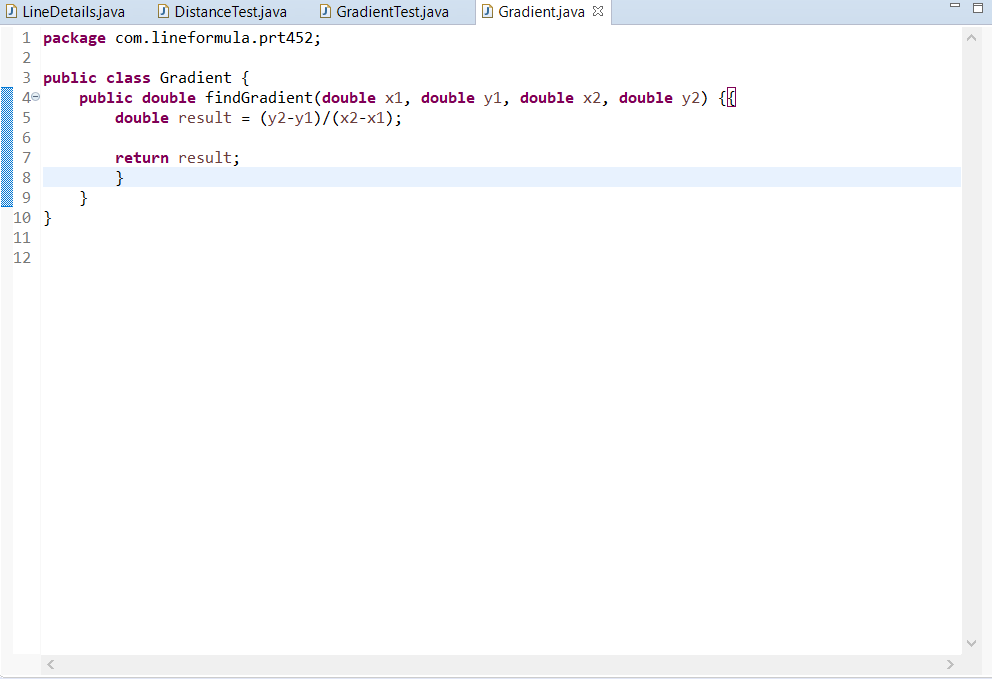
**Initial test before writing test cases (test failed)**



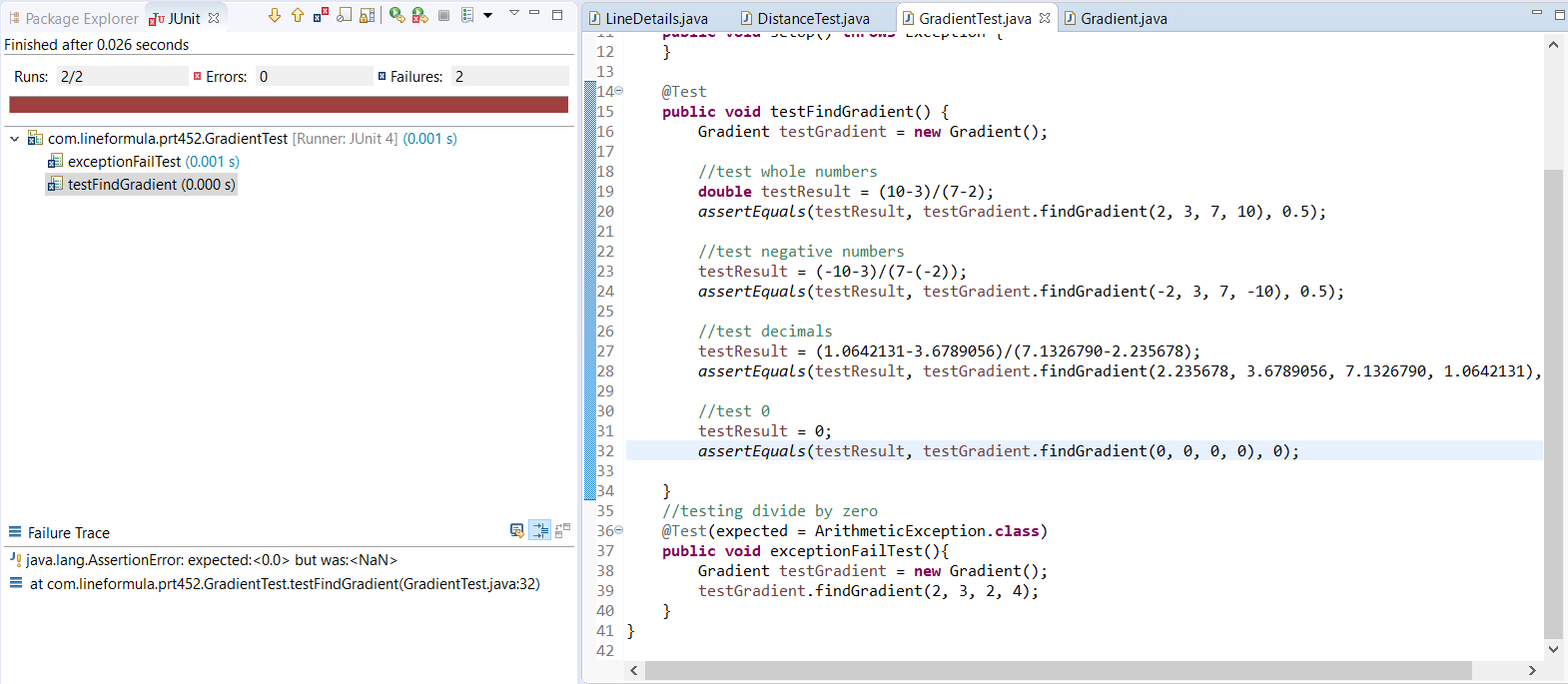
**Initial writing of test cases (tests failed since there is no implementation yet)**

****

**Initial Implementation code of Gradient class**

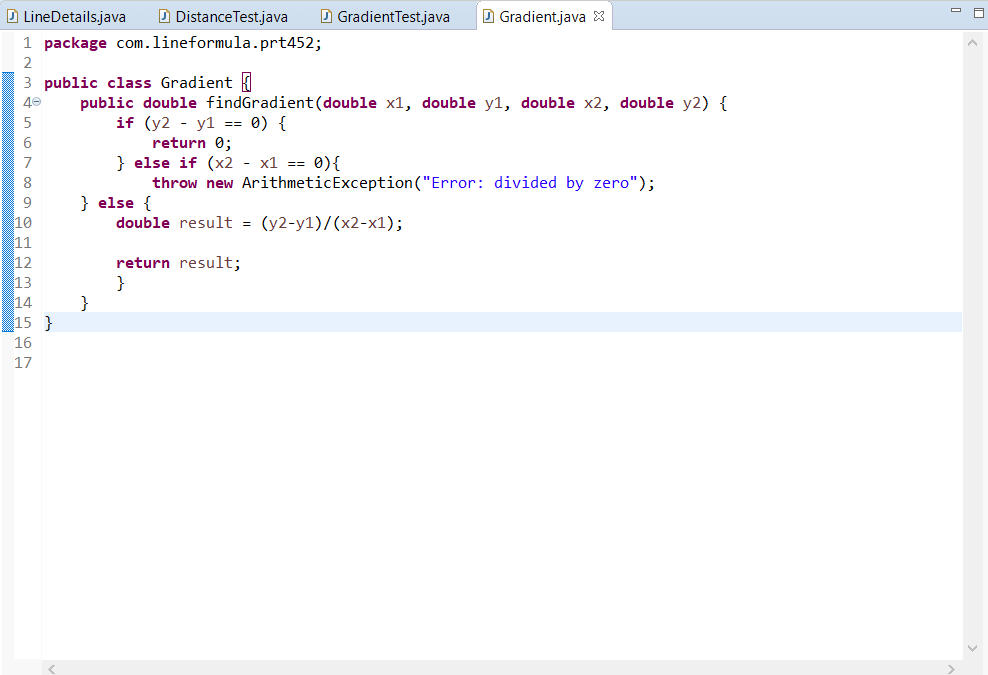
****

**Test case (last test failed, expected 0 but got NaN)**

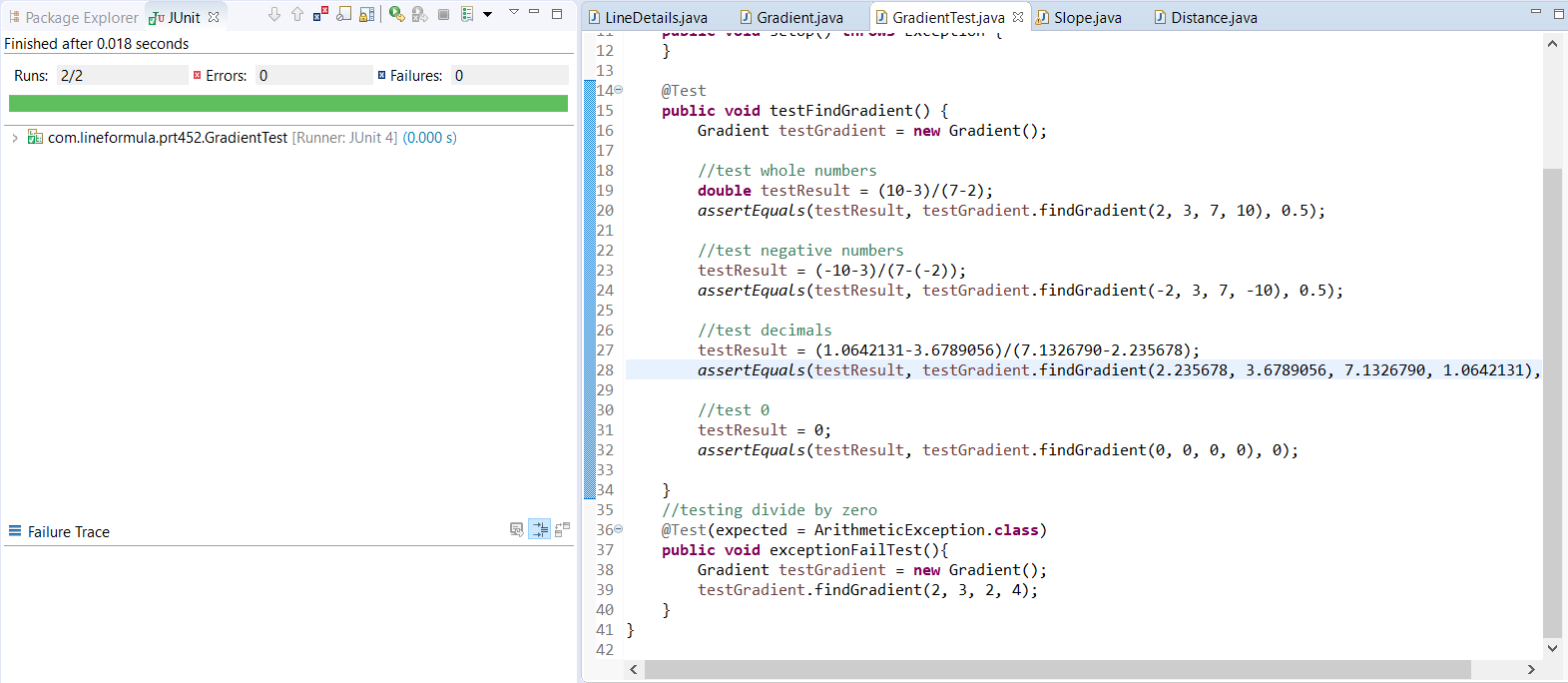


This test told me that I had I was getting a non-number instead of zero when all inputs were zero. I added two if statements to my code for the cases where y1 = y2 and x1 = x2. For the first statement, the method would be hard coded to return a result of zero, as when y1 = y2, the slope is zero. For the second statement, I made the program throw an exception when x1 = x2, as finding the slope would require dividing by zero which is impossible.

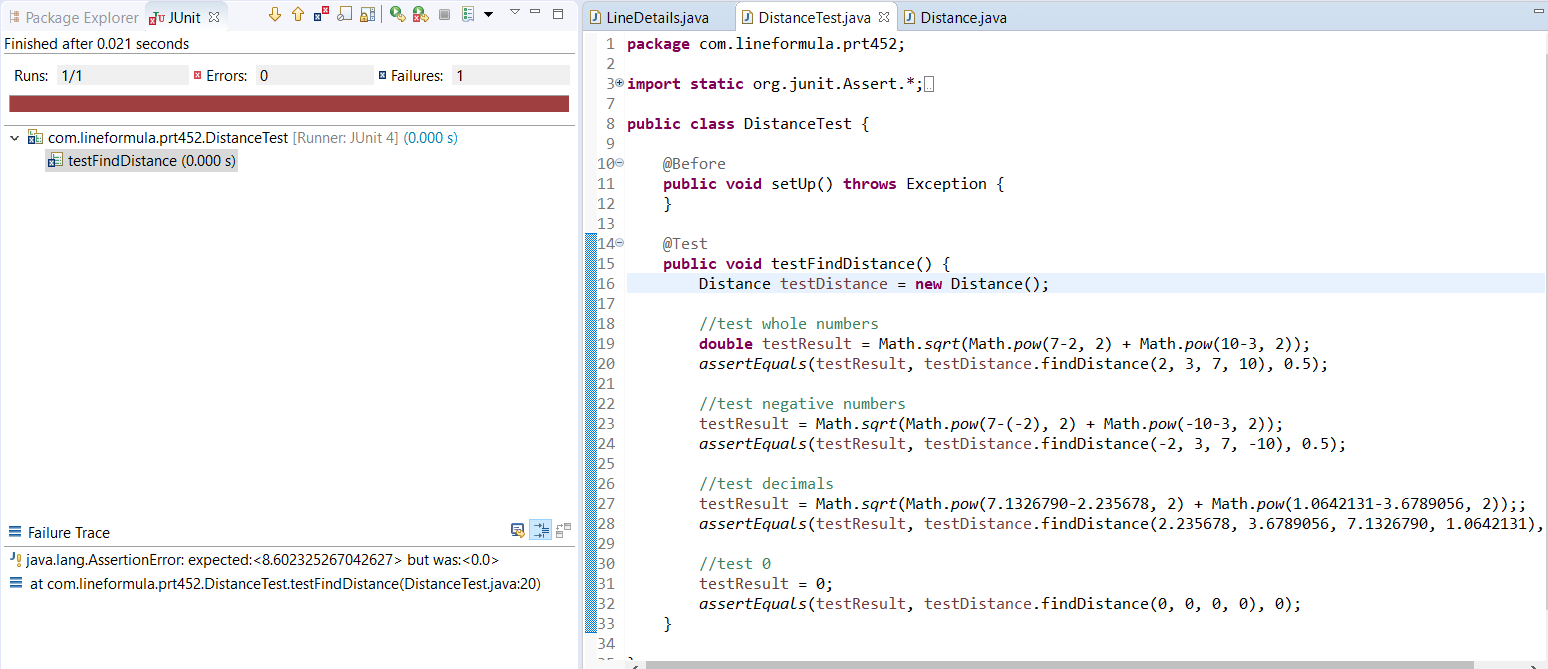
**Modified code**



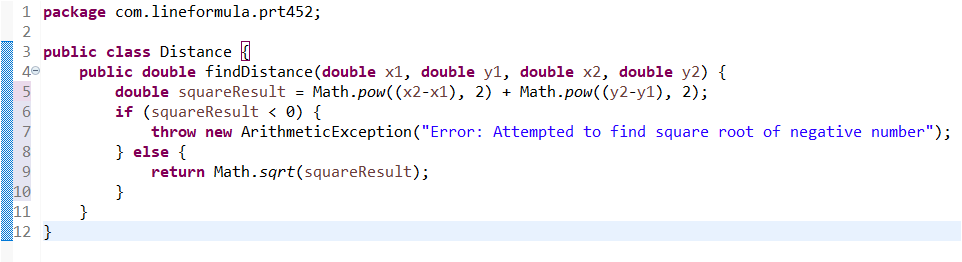
**Gradient class passing all tests**



**Initial Test Cases for Distance Class (test failed, no implementation)**

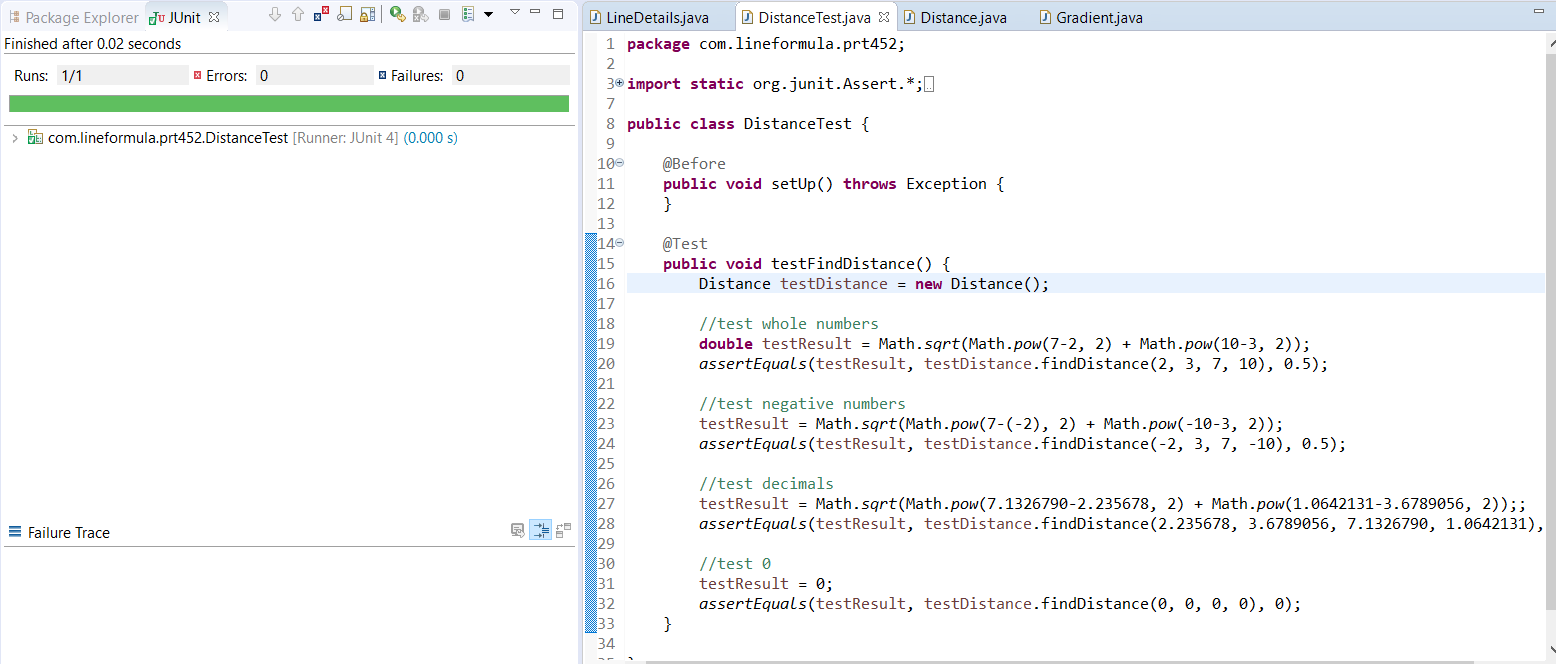
****

**Initial Implementation of Distance Code**

****

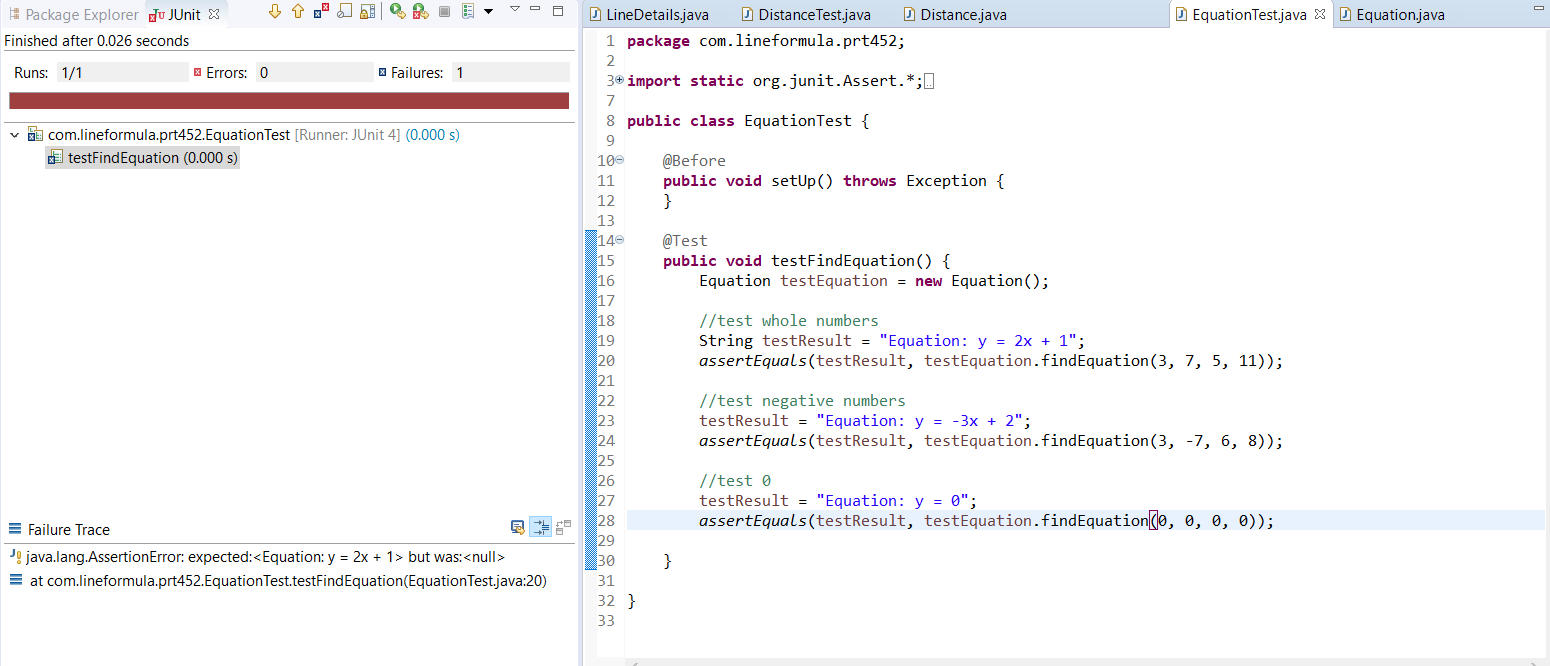
I included an if statement to throw an exception if the variable squareResult was negative, as this would try to find the square root of a negative number otherwise.

**Distance Class passing all tests**

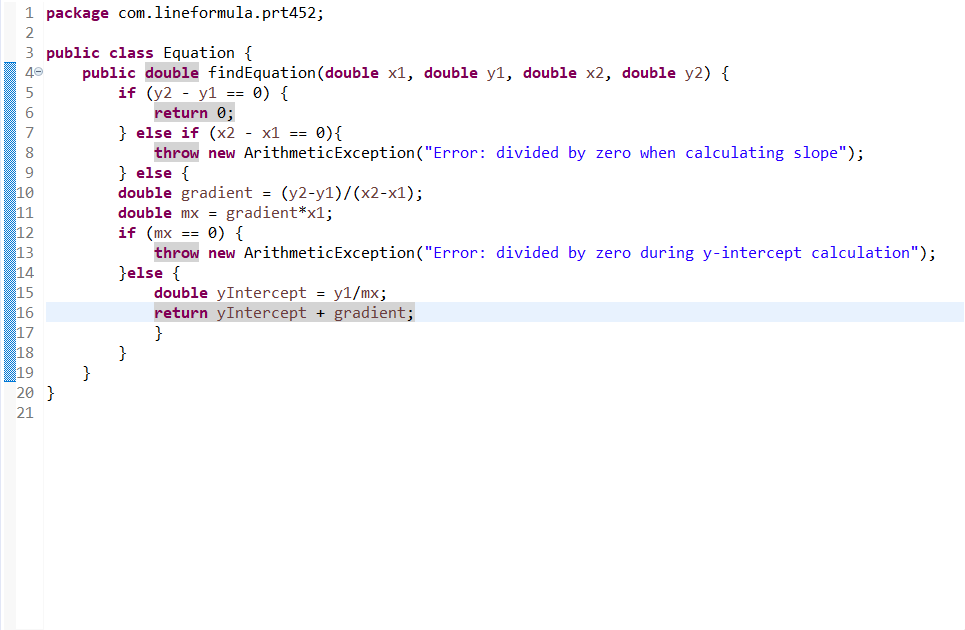
****

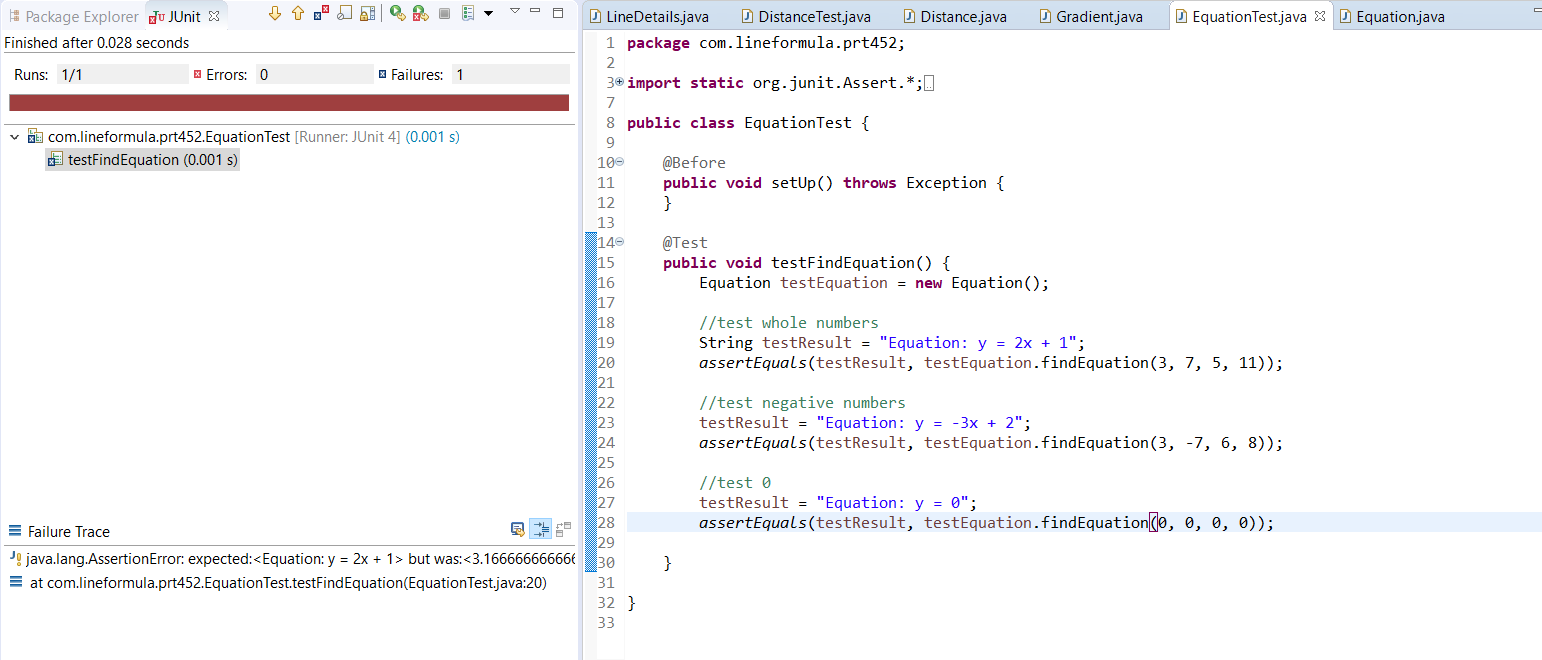
This functionality was relatively easy to implement as the operations required were simple (square roots and powers). The test passed without errors on my first run.

**Initial Test Cases for Equation Class (test failed, no implementation)**



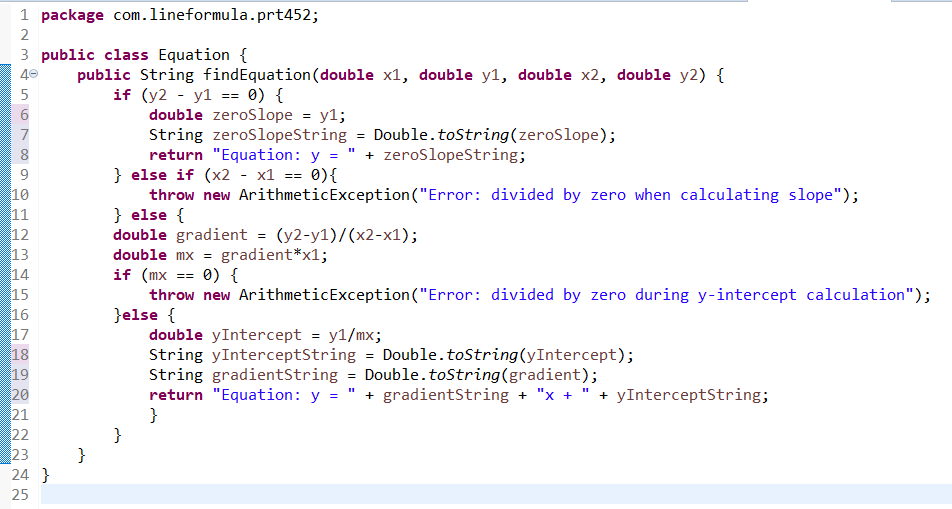
**Initial Implementation code for Equation class**

****

**Test case (test failed, incorrect return type)**

My code was returning a double type but I was looking for a string of the equation.

**Modified code**



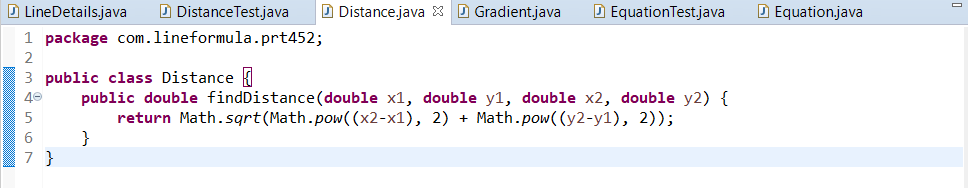
I modified the code to convert doubles to strings and return a string with the equation.

**Code Refactoring**

**Distance Class**

I reviewed the formula for finding the distance between two points and realised that it would be impossible to end up with a negative number before the square root process. This is because the formula is adding two squared numbers, and it is impossible to get a negative number when a number is squared, even if the original number is negative. Therefore, I can remove the negative number check without affecting the functionality. With this change, I can also remove the squareResult variable and have the class return the result of the operation directly.

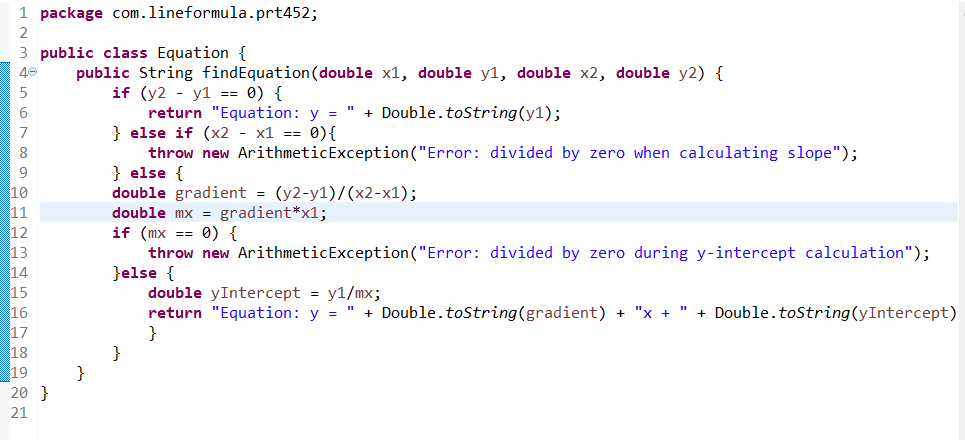
**Modified Code**



**Equation Class**

The Equation class is the longest class in my program, so it would be beneficial to reduce the size of it as much as possible to improve performance and readability. I removed the variables that stored the converted double to string values, and made the program return them directly.

**Modified Code**

****

**Part 2**

**Code smell can give indications that there is some issue with the codes and can be solved by refactoring. Identify 5 issues and their related code smells. Include solutions to fix up the issues.**

Five issues that can affect the quality of code, and their related code smells, are:

1. Having the same code structure in more than one place. This code smell is referred to as duplicated code, and can impact the maintenance of a project as a developer will have to search for each instance to fix any issues. This can be solved by extracting the repeated code and invoking the code from the needed places.
2. Having too many comments can detract from the readability of code. Comments are a type of code smell where there are more comments than necessary to explain each function. It is better to write the code in a way that humans can look at it and quickly figure out what it does, instead of having comments that break the reading flow of the program.
3. Having to make many small changes to different classes to modify the program is a code smell known as shotgun surgery. This is usually fixed by using Move Method or Move Field to group existing class behaviours together.
4. Classes with very little functional capability should be removed to improve code organisation. These classes are referred to as Lazy Classes, and can be fixed with Inline Class, as it moves features from one class to another. Subclasses with few functions can be combined with their superclass via Collapse Hierarchy.
5. Programmers may create classes, methods, fields, or parameters to be used in the future, but are never actually implemented. This code smell is referred to as Speculative Generality. Unused methods and classes can be removed via Inline Method and Inline Class respectively in order to merge these classes with more used ones. Unused fields can also be deleted from the program.

# References

Atwood, J., 2006. *Code Smells.* [Online]   
Available at: https://blog.codinghorror.com/code-smells/  
[Accessed 1 September 2018].

Mcfadyen, G., 2017. *Common Code Smells.* [Online]   
Available at: https://8thlight.com/blog/georgina-mcfadyen/2017/01/19/common-code-smells.html  
[Accessed 2 September 2018].

Refactoring Guru, 2018. *Code Smells.* [Online]   
Available at: https://refactoring.guru/refactoring/smells  
[Accessed 1 September 2018].