SE 317: Lab 4-a

Testing Boundary Conditions Using Exceptions Handling

Instructions

For additional details, use book Chapter 7: Page 79, 80, 81
Use the source code provided in the zip folder. There are six classes in the zip folders.

1. Bearing.java
2. BearingOutofRangeException.java
3. BearingTest.java
4. Rectangle.java
5. Rectangletest.java
6. ConstrainsSideTo.java

Lab objective: To understand the concepts of *throws* declaration and *try/catch* method.
Some classes have errors. You need to find and fix the errors, then submit the screenshots of the corrected code by using two different methods: "throws" method and "try/catch method". This assignment will also help understand how you can do unit testing at different boundaries and how "range" works.

Steps:

1- Run the BearingTest.Jav code
2- You will get error messages as in figure 1 below
3- Inspect the BearingTest.java, it has 3 functions:
   i. `public void answersValidBearing()`
   ii. `public void answersAngleBetweenItAndAnotherBearing()`
   iii. `public void angleBetweenIsNegativeWhenThisBearingSmaller()`

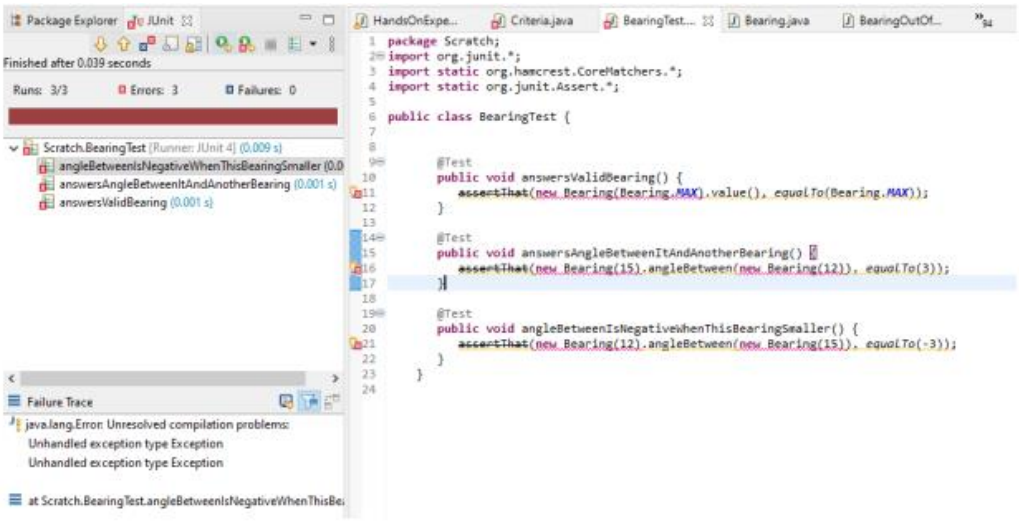


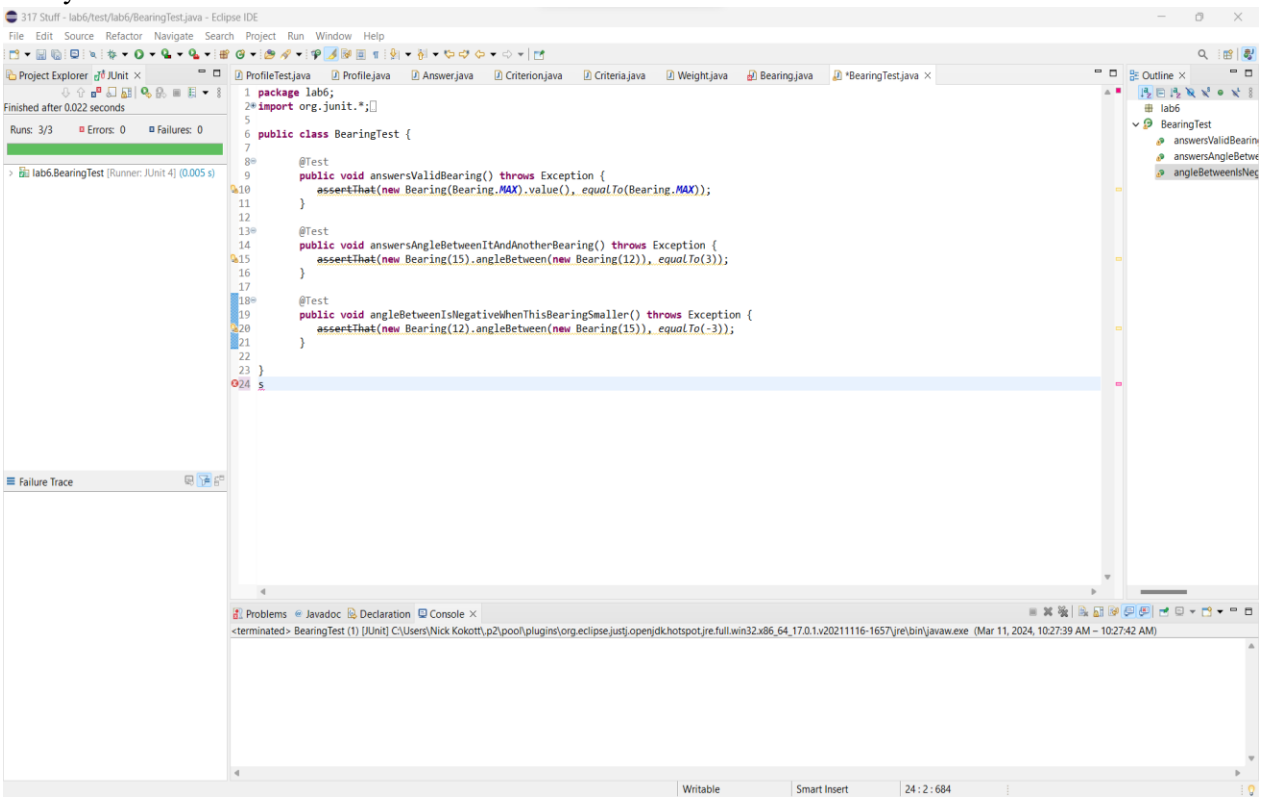
Fig 1

1. **TODO:**

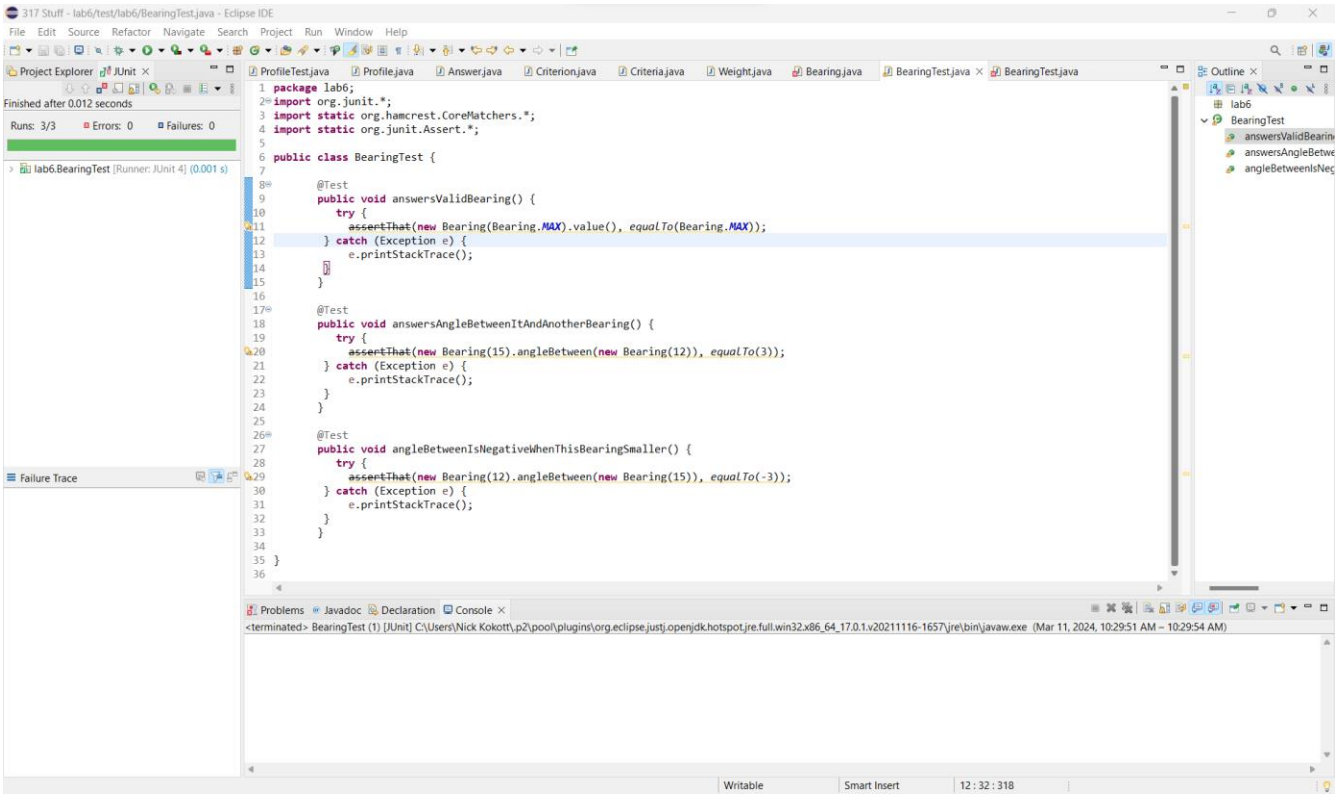
Part 1

These 3 functions contain some errors. You need to fix them by using both **throws** method and **try/catch** method.

1- First, use **throws** method to fix the code. When you finish, take the screenshot of the passed result with your code.



2- Next, Replace the throw exception with the "**try/catch**" method, run the code again and submit the screenshot of the **passed** result with your code



In both cases, when you take the screenshots, make sure you also take screenshot of the BearingTest.java code so that we can see your code.

Part 2

1- After fixing the code, inspect the answersAngleBetweenItAndAnotherBearing() function and the `angleBetweenIsNegativeWhenThisBearingSmaller().`

Analyze the code in Bearing.java.

Note: A circle has 360 degrees in either direction (clockwise or counter clockwise). Rather than storing the direction of a travel as a native type, Bearing.java encapsulates the direction along with logic to constrain its range.

2. **TODO:**

Write 8 test cases similar to angleBetweenIsNegativeWhenThisBearingSmaller() functions and use try/catch method or throws function (either one) and make sure the test cases pass. Take a screenshot of the test cases. Your test cases should test different bearings (0, 355, 90, 55, 100, 12, 123, etc.)

Hint: Inspect bearing.java code to see how it works. Create similar test cases and take a screenshot of test cases and make sure it passes.

Example:-
Start with similar test case of angleBetweenIsNegativeWhenThisBearingSmaller() function.
Note, this example uses Throws Method but you can use any method. See below.

```
@Test
public void angleBetweenIsNegativeWhenThisBearingSmaller2() throws Exception
{
	assertThat(new Bearing(5).angleBetween(new Bearing(15)), equalTo(-10));
```

}

Note that `angleBetween()` returns an int. We are not placing any range restrictions on the result.

Part 3

Inspect the classes Rectangle and RectangleTest from Lab 4 zip folder

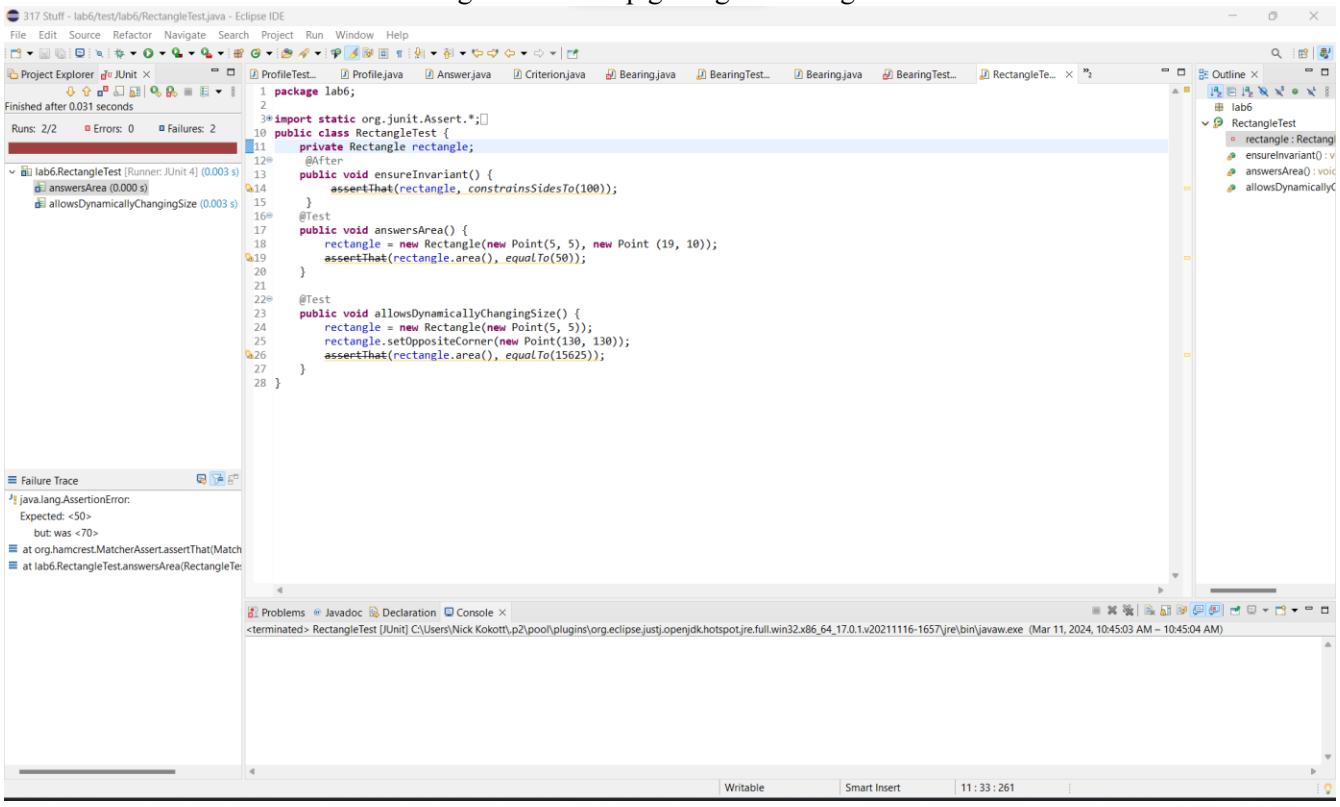
Some constraints might not be as straightforward. Suppose we have a class that maintains two points, each point is an (x, y) integer tuple. The **constraint** on the range is that the two points must describe a **rectangle** with no side greater than 100 units. That is, the allowed range of values for both x, y pairs is interdependent.

We want a range assertion for any behavior that can affect a coordinate, to ensure that the resulting range of the x, y pairs remains legitimate—that the *invariant* on the Rectangle holds true.

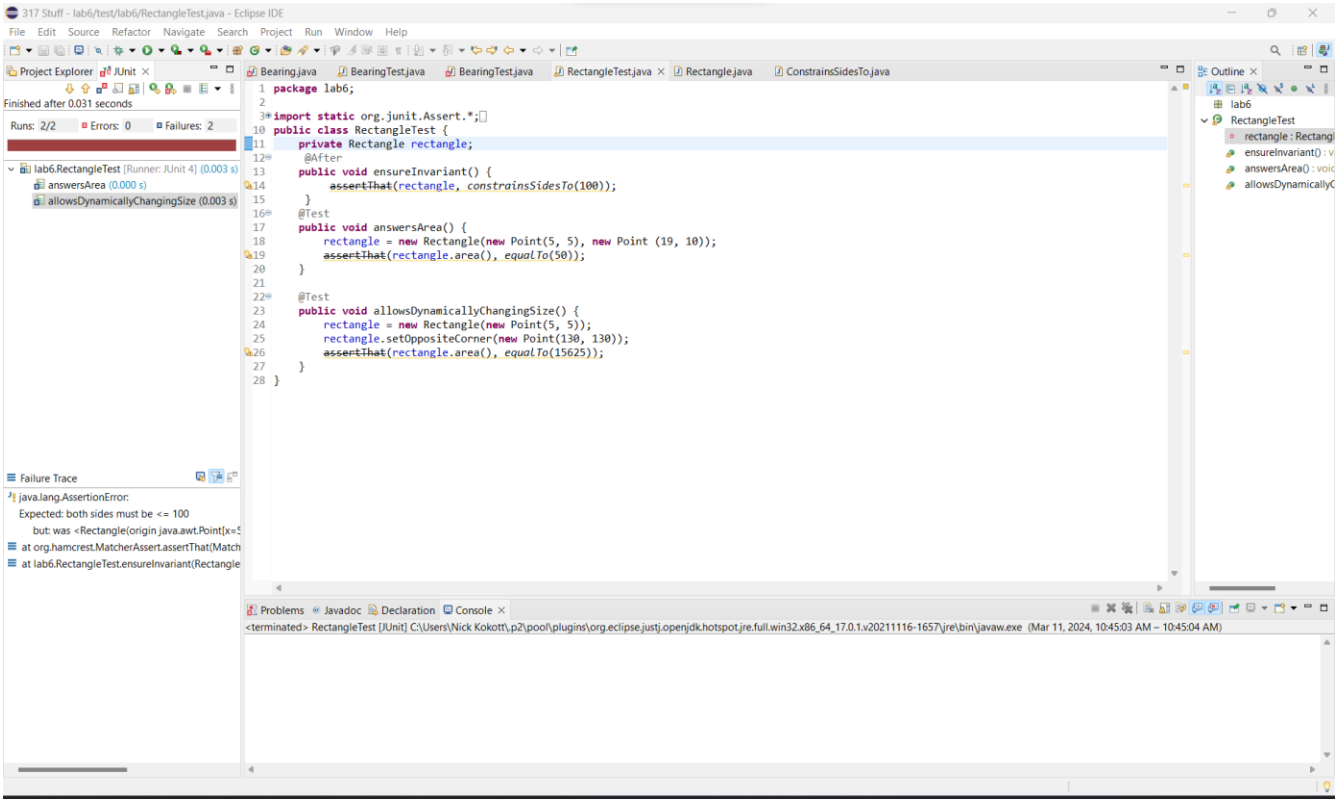
More formally: an **invariant** is a condition that holds true throughout the execution of some chunk of code. In this case, we want the invariant to hold true for the lifetime of the Rectangle object—that is, any time its state changes.

We can add invariants, in the form of assertions, to the @After method so that they run upon completion of any test. An implementation for the invariant for our constrained Rectangle class looks like RectangleTest in the source code folder.

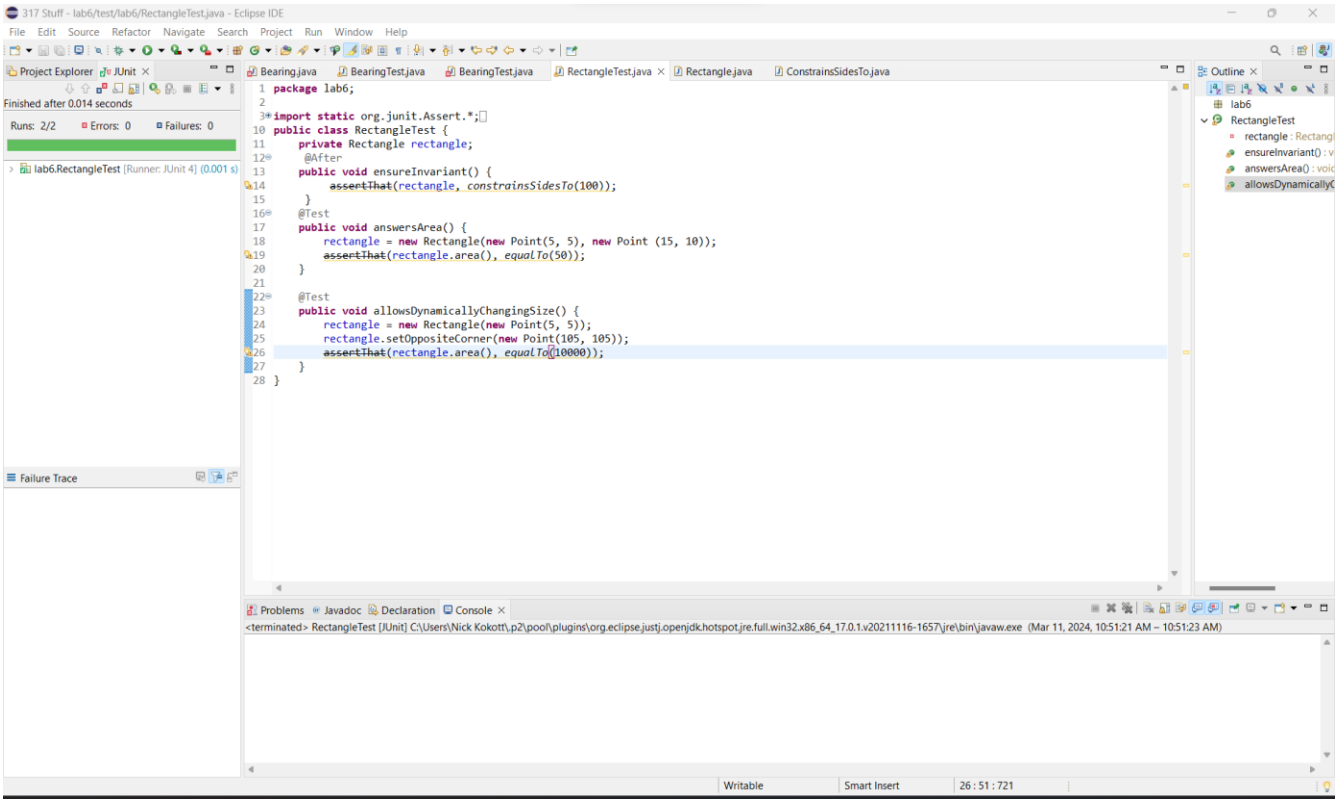
3. **TODO:**
   Run the test cases in RectangleTest.Java.
   Any error? Take a screenshot of your code output
   There were two errors in this RectangleTest class, one per test method.
   The first calculates the area of a rectangle and ends up getting the wrong result.



The second causes an error due to the sides being too large. There is a natural constraint that the sides must be <= 100.



4. **TODO:** Fix the error(s) of the code and run the code again.
   Take a screenshot of your code and output
   Hint: Inspect Rectangle.java and look at the function public int area(). Analyze it.



5. **TODO:**

Answer the following questions:

1. What is throw exception and how does it fix the code?
   A throw exception is in essence an indicator to the program that this method may throw some sort of exception upon execution. This allows the code when not working to pass upon any exception that might be thrown during runtime.
2. What is try-catch method and how does it fix the code
   A try-catch method is a different way to handle exceptions that has the program attempt to run the method first, if the method ends up returning an exception the catch block will see this and know that this may have been supposed to happen. This wil allow the code when failing to end up succeeding when this catch block ends up being called to if an exemption is returned.
3. Is there any difference between throw exception and try-catch method? If yes, explain.
   There is a difference between the two however, it is very mild. In a throw method, the throw immediately lets the program know that there will potentially be an exception and to be ready for it if it happens. In a try-catch method, the program is unaware of any exception until it ends up being called and then referred to in the catch block.