

SE 317: Lab 6

Instructions

Boundary Conditions: The Correct Way

Yuichi Hamamoto

Book Pages (for additional description): Chapter 7: Page 79, 80, 81

Use the source code provided in the zip folder. There are six classes in the zip folder.

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.Java 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()`

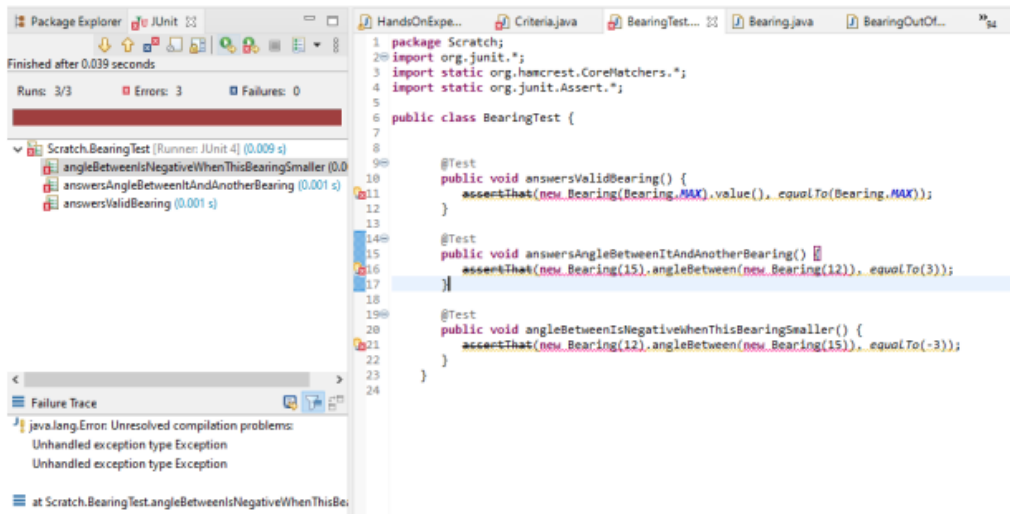


Fig 1

TODO:

Part 1

These 3 functions contain some errors. You need to fix the 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.



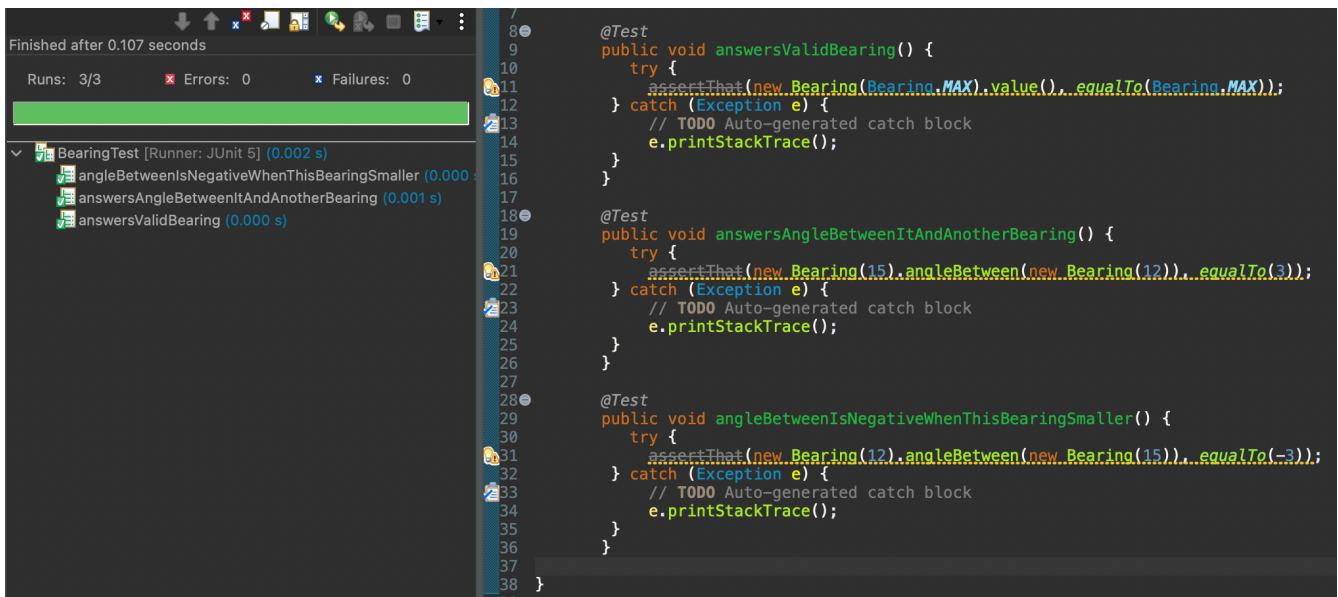
```
1 package lab6;
2 import org.junit.*;
3
4
5
6 public class BearingTest {
7
8     @Test
9     public void answersValidBearing() throws Exception {
10         assertEquals(new Bearing(Bearing.MAX).value(), new Bearing(MAX));
11     }
12
13     @Test
14     public void answersAngleBetweenItAndAnotherBearing() throws Exception {
15         assertEquals(new Bearing(15).angleBetween(new Bearing(12)), new Bearing(3));
16     }
17
18     @Test
19     public void angleBetweenIsNegativeWhenThisBearingSmaller() throws Exception {
20         assertEquals(new Bearing(12).angleBetween(new Bearing(15)), new Bearing(-3));
21     }
22
23 }
24
```

Finished after 0.124 seconds

Runs: 3/3 Errors: 0 Failures: 0

> BearingTest [Runner: JUnit 5] (0.001 s)

- 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



```
7
8
9     @Test
10     public void answersValidBearing() {
11         try {
12             assertEquals(new Bearing(Bearing.MAX).value(), new Bearing(MAX));
13         } catch (Exception e) {
14             // TODO Auto-generated catch block
15             e.printStackTrace();
16         }
17
18     @Test
19     public void answersAngleBetweenItAndAnotherBearing() {
20         try {
21             assertEquals(new Bearing(15).angleBetween(new Bearing(12)), new Bearing(3));
22         } catch (Exception e) {
23             // TODO Auto-generated catch block
24             e.printStackTrace();
25         }
26     }
27
28     @Test
29     public void angleBetweenIsNegativeWhenThisBearingSmaller() {
30         try {
31             assertEquals(new Bearing(12).angleBetween(new Bearing(15)), new Bearing(-3));
32         } catch (Exception e) {
33             // TODO Auto-generated catch block
34             e.printStackTrace();
35         }
36     }
37
38 }
39
```

Finished after 0.107 seconds

Runs: 3/3 Errors: 0 Failures: 0

✓ BearingTest [Runner: JUnit 5] (0.002 s)

- ✓ angleBetweenIsNegativeWhenThisBearingSmaller (0.000 s)
- ✓ answersAngleBetweenItAndAnotherBearing (0.001 s)
- ✓ answersValidBearing (0.000 s)

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.

1. 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.

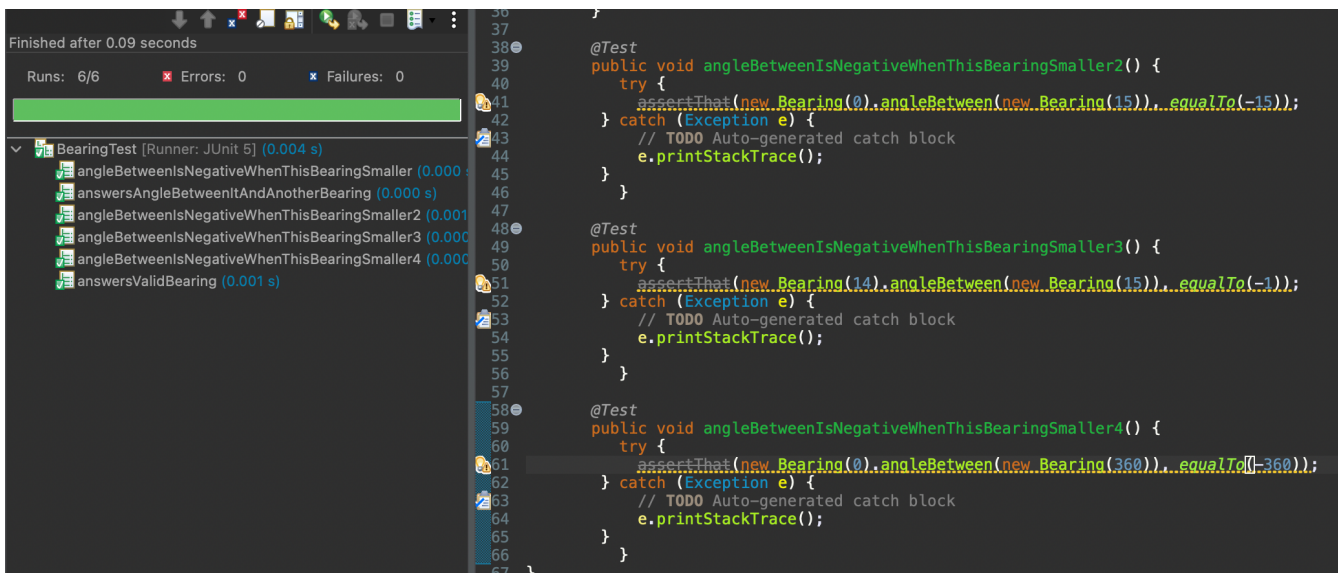
Example:-

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.



The screenshot shows an IDE with a test runner on the left and source code on the right. The test runner shows 'Finished after 0.09 seconds', 'Runs: 6/6', 'Errors: 0', and 'Failures: 0'. Below this, a list of test cases for 'BearingTest' is shown, all with green checkmarks indicating they passed:

- angleBetweenIsNegativeWhenThisBearingSmaller (0.000 s)
- answersAngleBetweenItAndAnotherBearing (0.000 s)
- angleBetweenIsNegativeWhenThisBearingSmaller2 (0.001 s)
- angleBetweenIsNegativeWhenThisBearingSmaller3 (0.000 s)
- angleBetweenIsNegativeWhenThisBearingSmaller4 (0.000 s)
- answersValidBearing (0.001 s)

The source code on the right shows the implementation of these test cases:

```
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67

@Test
public void angleBetweenIsNegativeWhenThisBearingSmaller2() {
    try {
        assertThat(new Bearing(0).angleBetween(new Bearing(15)), equalTo(-15));
    } catch (Exception e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
}

@Test
public void angleBetweenIsNegativeWhenThisBearingSmaller3() {
    try {
        assertThat(new Bearing(14).angleBetween(new Bearing(15)), equalTo(-1));
    } catch (Exception e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
}

@Test
public void angleBetweenIsNegativeWhenThisBearingSmaller4() {
    try {
        assertThat(new Bearing(0).angleBetween(new Bearing(360)), equalTo(-360));
    } catch (Exception e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
}
```

Part 3

Inspect the classes `Rectangle` and `RectangleTest` from Lab 6 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.

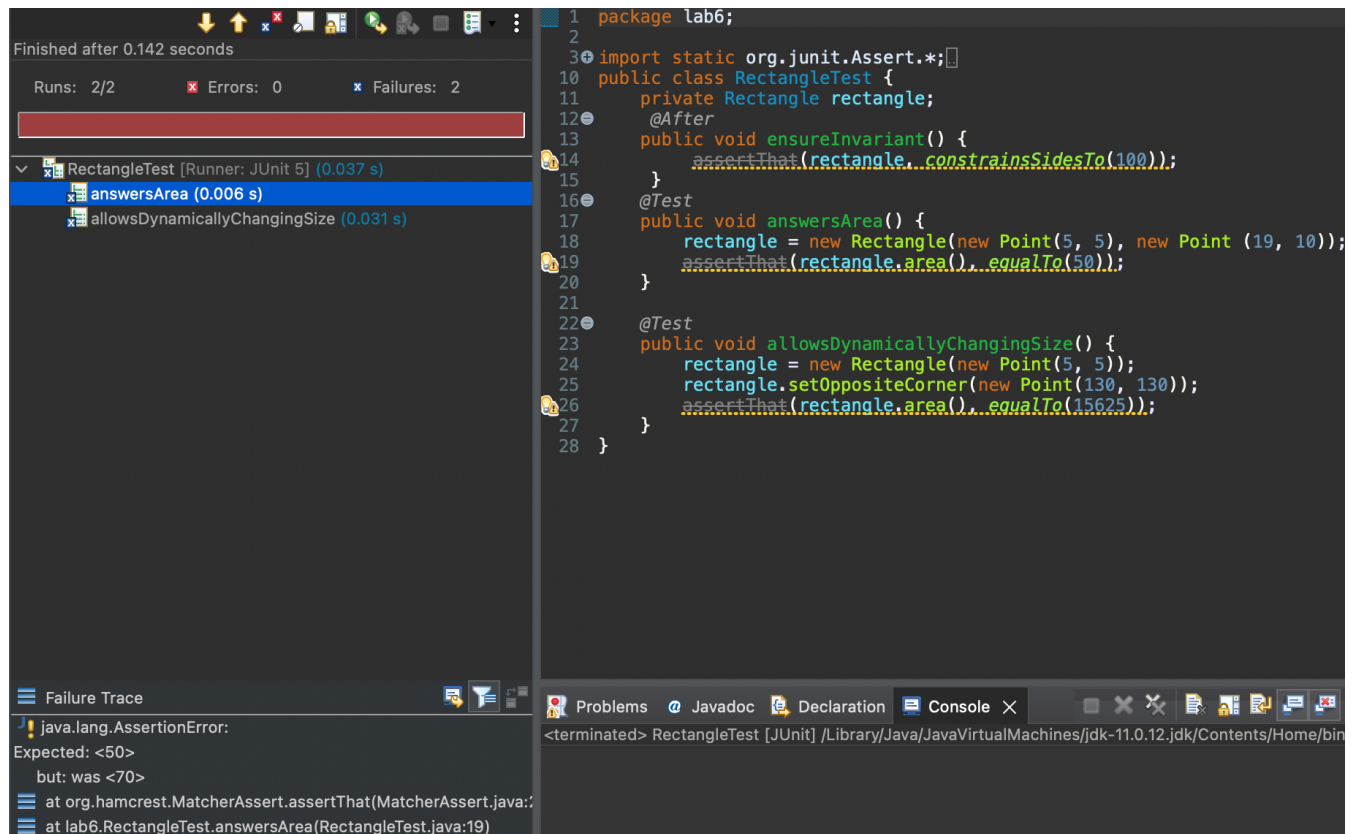
More formally: an **invariant** is a condition that holds true throughout the execution of some chunk of code. In this case, we want the invariant to hold true for the lifetime of the `Rectangle` object—that is, any time its state changes.

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.

2. TODO:

Run the test cases in `RectangleTest.java`.

Any error? Take a screenshot of your code output



The screenshot shows an IDE with the following components:

- Test Results Panel (Left):** Shows the test run summary: "Finished after 0.142 seconds", "Runs: 2/2", "Errors: 0", "Failures: 2". The test suite is "RectangleTest [Runner: JUnit 5] (0.037 s)". Two tests are listed: "answersArea (0.006 s)" and "allowsDynamicallyChangingSize (0.031 s)". Both tests are marked as failed with red 'x' icons.
- Source Code Editor (Right):** Displays the code for `RectangleTest.java`. The code includes:
 - Package declaration: `package lab6;`
 - Imports: `import static org.junit.Assert.*;`
 - Class definition: `public class RectangleTest {`
 - Private field: `private Rectangle rectangle;`
 - `@After` method: `public void ensureInvariant() {`
 - Assertion: `assertThat(rectangle, constraintsSidesTo(100));`
 - Closing brace: `}`
 - `@Test` method: `public void answersArea() {`
 - Setup: `rectangle = new Rectangle(new Point(5, 5), new Point(19, 10));`
 - Assertion: `assertThat(rectangle.area(), equalTo(50));`
 - Closing brace: `}`
 - `@Test` method: `public void allowsDynamicallyChangingSize() {`
 - Setup: `rectangle = new Rectangle(new Point(5, 5));`
 - Action: `rectangle.setOppositeCorner(new Point(130, 130));`
 - Assertion: `assertThat(rectangle.area(), equalTo(15625));`
 - Closing brace: `}`
 - Closing brace for class: `}`
- Failure Trace (Bottom Left):** Shows the error for the `answersArea` test:
 - Exception: `java.lang.AssertionError:`
 - Expected: `<50>`
 - but: was `<70>`
 - Stack trace:
 - `at org.hamcrest.MatcherAssert.assertThat(MatcherAssert.java:20)`
 - `at lab6.RectangleTest.answersArea(RectangleTest.java:19)`
- Problems Panel (Bottom Right):** Shows the message: `<terminated> RectangleTest [JUnit] /Library/Java/JavaVirtualMachines/jdk-11.0.12.jdk/Contents/Home/bin`

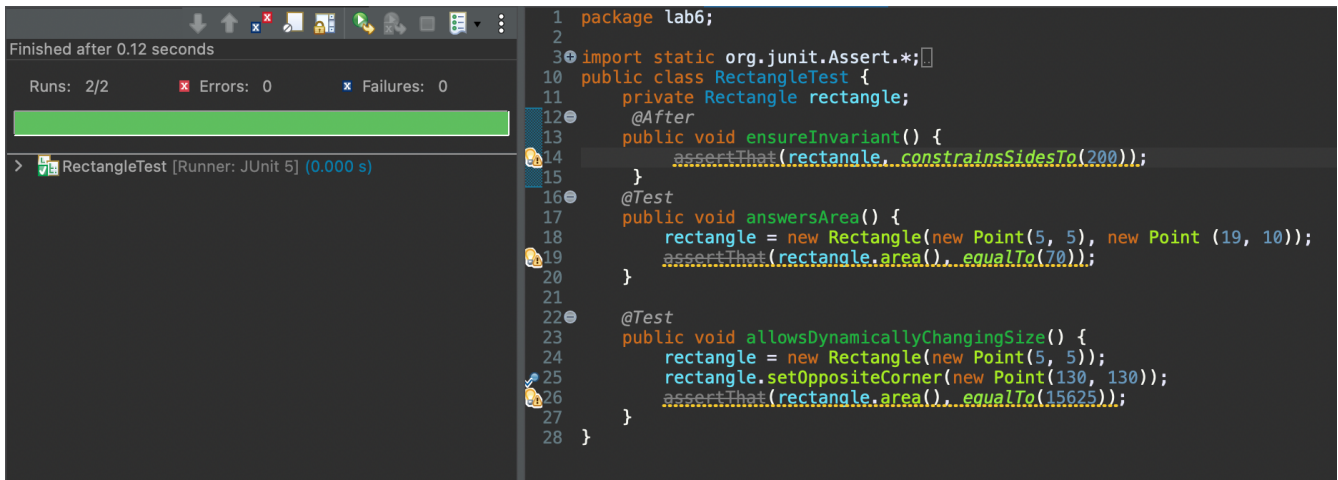
3. **TODO:** Fix the error(s) of the code and run the code again.

Take a screenshot of your code and output

Hint: Inspect Rectangle.java and look at the function public int area(). Analyze it.

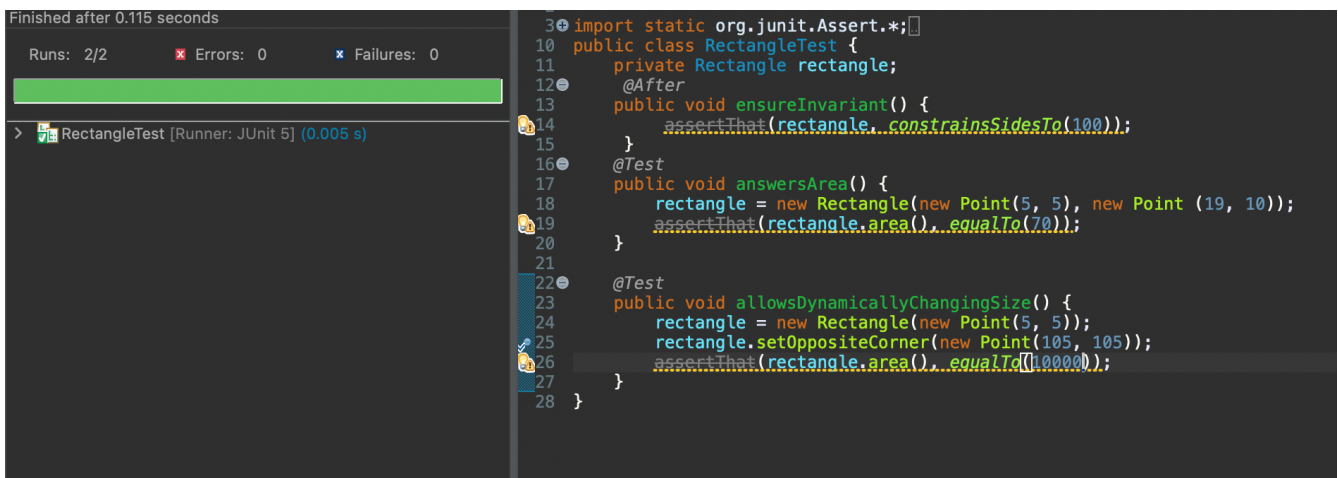
Here we can solve the issue by changing the constrain or make the value within the constrain

1: change the constrain



```
1 package lab6;
2
3 import static org.junit.Assert.*;
4
10 public class RectangleTest {
11     private Rectangle rectangle;
12     @After
13     public void ensureInvariant() {
14         assertThat(rectangle.constrainsSidesTo(200));
15     }
16     @Test
17     public void answersArea() {
18         rectangle = new Rectangle(new Point(5, 5), new Point(19, 10));
19         assertThat(rectangle.area(), equalTo(70));
20     }
21
22     @Test
23     public void allowsDynamicallyChangingSize() {
24         rectangle = new Rectangle(new Point(5, 5));
25         rectangle.setOppositeCorner(new Point(130, 130));
26         assertThat(rectangle.area(), equalTo(15625));
27     }
28 }
```

2: change the value



```
3 import static org.junit.Assert.*;
4
10 public class RectangleTest {
11     private Rectangle rectangle;
12     @After
13     public void ensureInvariant() {
14         assertThat(rectangle.constrainsSidesTo(200));
15         assertThat(rectangle.constrainsSidesTo(100));
16     }
17     @Test
18     public void answersArea() {
19         rectangle = new Rectangle(new Point(5, 5), new Point(19, 10));
20         assertThat(rectangle.area(), equalTo(70));
21     }
22
23     @Test
24     public void allowsDynamicallyChangingSize() {
25         rectangle = new Rectangle(new Point(5, 5));
26         rectangle.setOppositeCorner(new Point(105, 105));
27         assertThat(rectangle.area(), equalTo(11025));
28     }
29 }
```

4. **TODO:**

Answer the following questions:

1. What is throw exception and how does it fix the code?
It throws an exception when they find an exception from the method. It allows us to know what was wrong in detail and help us fix the code.
2. What is try-catch method and how does it fix the code
Try-catch is similar to throw exception, but instead of throwing, it catches the exception and allows us to handle the exception by ourselves.
3. Is there any difference between throw exception and try-catch method? If yes, explain.
As I answered in 2, the difference between them is either it throws the exception or catches the exception.