Rectangle Test Report

Team members:

Mario LoPrinzi
&
Christina Xuan Yu

1. **Test Plan:**

- Test all the methods, to see if they function as described.
- We focus on some extreme cases, such as testing with negative numbers, randomly generated numbers, numbers that are out of bounds.
- Make sure the program fails where it should.
    - Make sure a Rectangle cannot be made with negative width or height
    - Make sure the area cannot overflow an integer
    - Make sure a Rectangle with no area cannot contain any points

2. **Positive Test: Christina Yu: 5, Mario LoPrinzi: 3**

- testArea(): test rectangle area starting from random point with random width and height
- testDiagonal(): test diagonal using random width and height
- testUnion(): test the rectangle union with empty
- testIntersects(): test if the rectangle intersects with itself
- testIntersection(): test if a random rectangle intersection with itself
- testEquals(): Now checks for equality in the negative quadrant as well.
- testToString(): Check to see if negative quadrant integers are printing as expected.
- testContains(): Make sure a zero area rectangle is working as expected.

3. **Negative Test: Christina Yu & Mario LoPrinzi: 2**

- testInit(): the Rectangle's width and height should not be negative
- testArea(): Result is incorrect as the int value is overflowed

4. **Fixed Incorrect Test Method: Christina Yu: 1**

- testDiagonal()

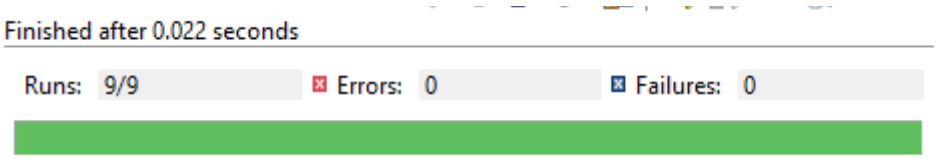
5. **Fixed Incorrect Code in Original Project: Mario LoPrinzi : 3**

- Throw exception in Rectangle constructor for negative width and height
- Throw exception in Rectangle area for integer overflow

- Contains() returns false for rectangles that have no area to contain things in.

6. **Screen Shots of Test Results:**

- TestArea() (Positive & Negative test) & Throw an exception in area():



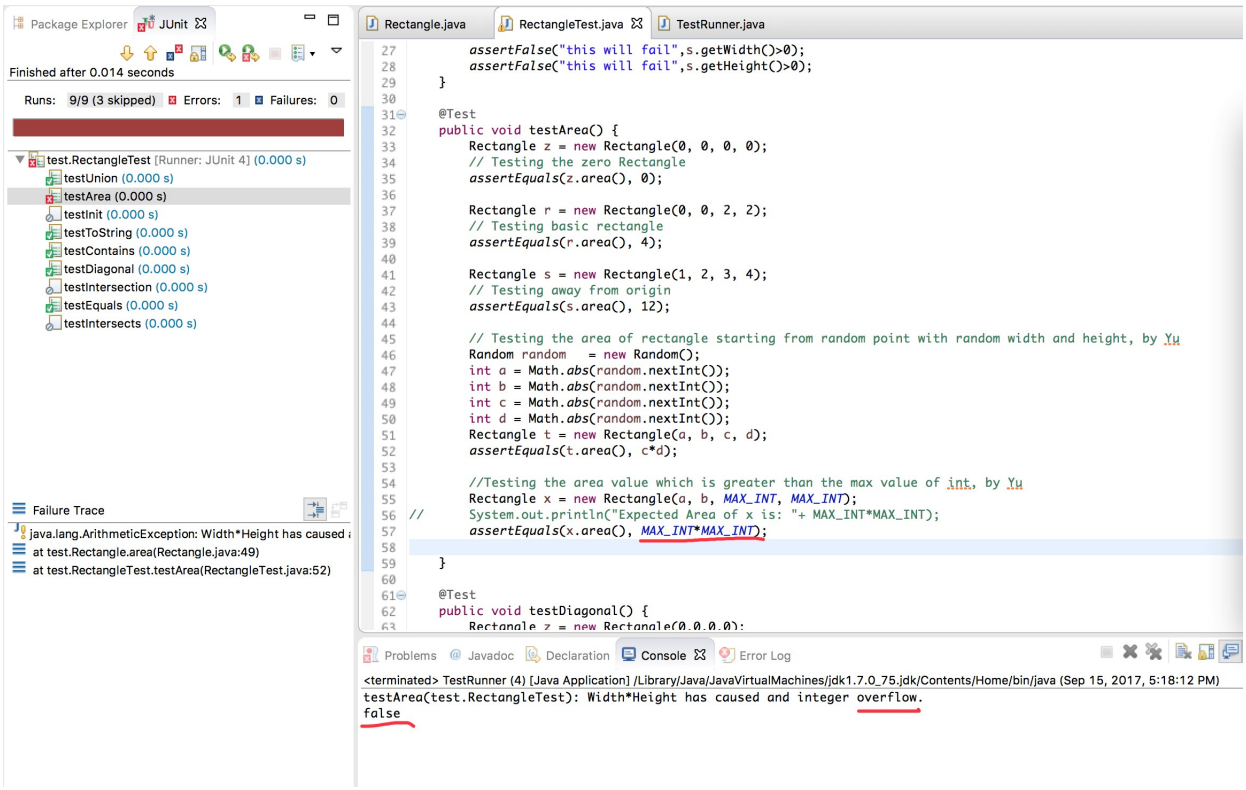

```
// Accessors for computed values
public int area() {
    // This checks for integer overflow from the multiplication.
    // Mario LoPrinzi
    if (width * height < 0) {
        throw new ArithmeticException("Width*Height has caused and integer overflow.");
    }
    return width * height;
}
```

```java
@Test
public void testArea() {
    Rectangle z = new Rectangle(0, 0, 0, 0);
    // Testing the zero Rectangle
    assertEquals(z.area(), 0);

    Rectangle r = new Rectangle(0, 0, 2, 2);
    // Testing basic rectangle
    assertEquals(r.area(), 4);

    Rectangle s = new Rectangle(1, 2, 3, 4);
    // Testing away from origin
    assertEquals(s.area(), 12);

    // Testing the area of rectangle starting from random point with random width
    // and height, by Yu
    Random random = new Random();
    int a = random.nextInt(10000);
    int b = random.nextInt(10000);
    int c = Math.abs(random.nextInt(10000));
    int d = Math.abs(random.nextInt(10000));
    Rectangle t = new Rectangle(a, b, c, d);
    assertEquals(t.area(), c * d);

    // Testing the area value which is greater than the max value of int, by Yu
    Rectangle x = new Rectangle(a, b, (int) Math.pow(2, 32), (int) Math.pow(2, 32));
    // System.out.println("Expected Area of x is: "+ MAX_INT*MAX_INT);

    // throws exception if area turns negative from integer overflow. Mario LoPrinzi
    try {
        x.area();
    } catch (ArithmeticException error) {
        assertEquals("Width*Height has caused and integer overflow.", error.getMessage());
    }
```

We generate random numbers to test rectangle area starting from random point with random width and height, as it may cover more situations.

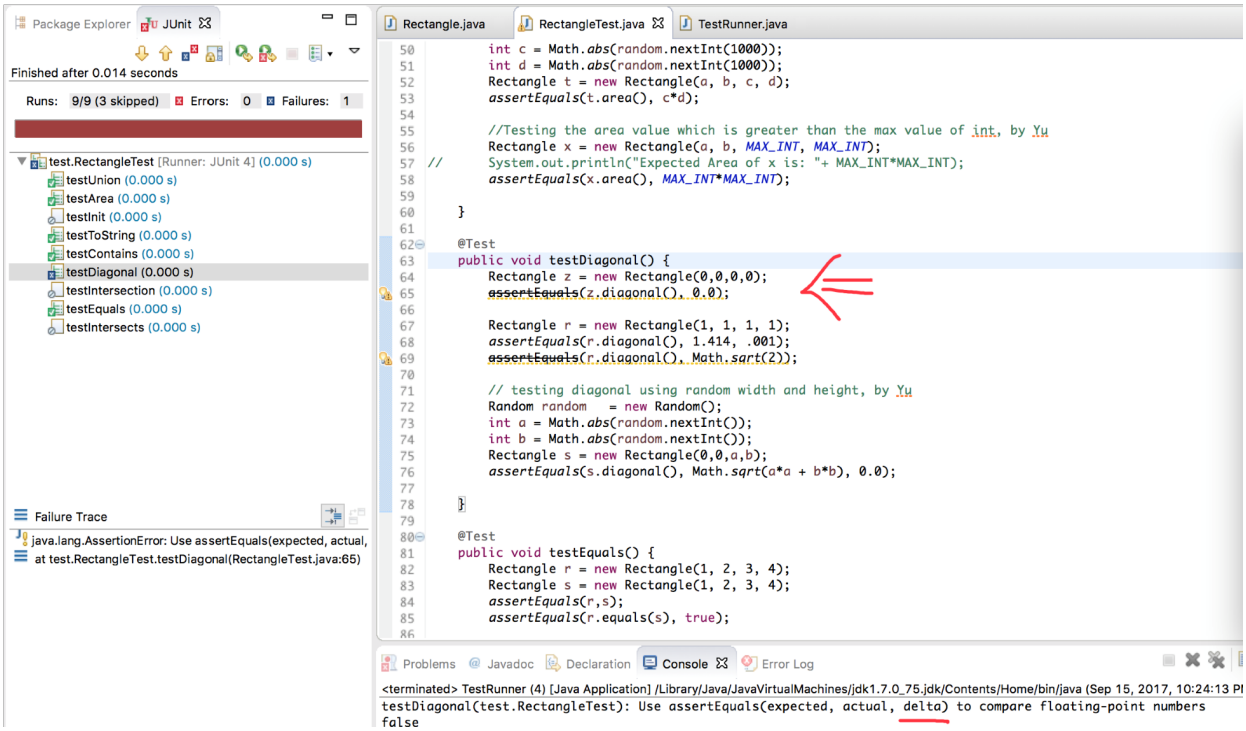
We wrote a negative test to check the area of the rectangle with the width and height both be MAX_INT— 2^31-1. Overflow happened, and the result of area become 1.

Then we threw an exception in Rectangle area() to catch the error above. This time we used the same negative test, and the program caught the overflow error.

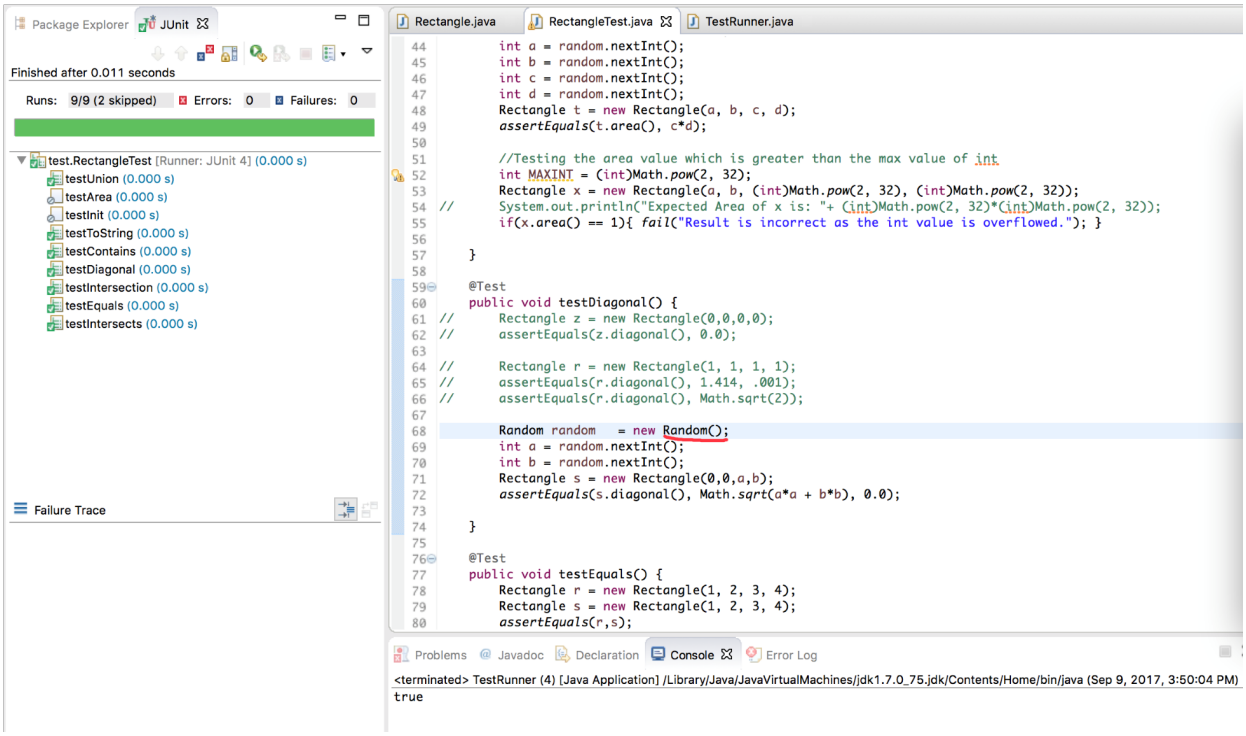


- TestDiagonal() & Fixed Incorrect Test Method

  The TestDiagonal() code has an error with a parameter missing in the assertEquals(), and the test file could not be compiled. We fixed the error by adding the correct delta value.

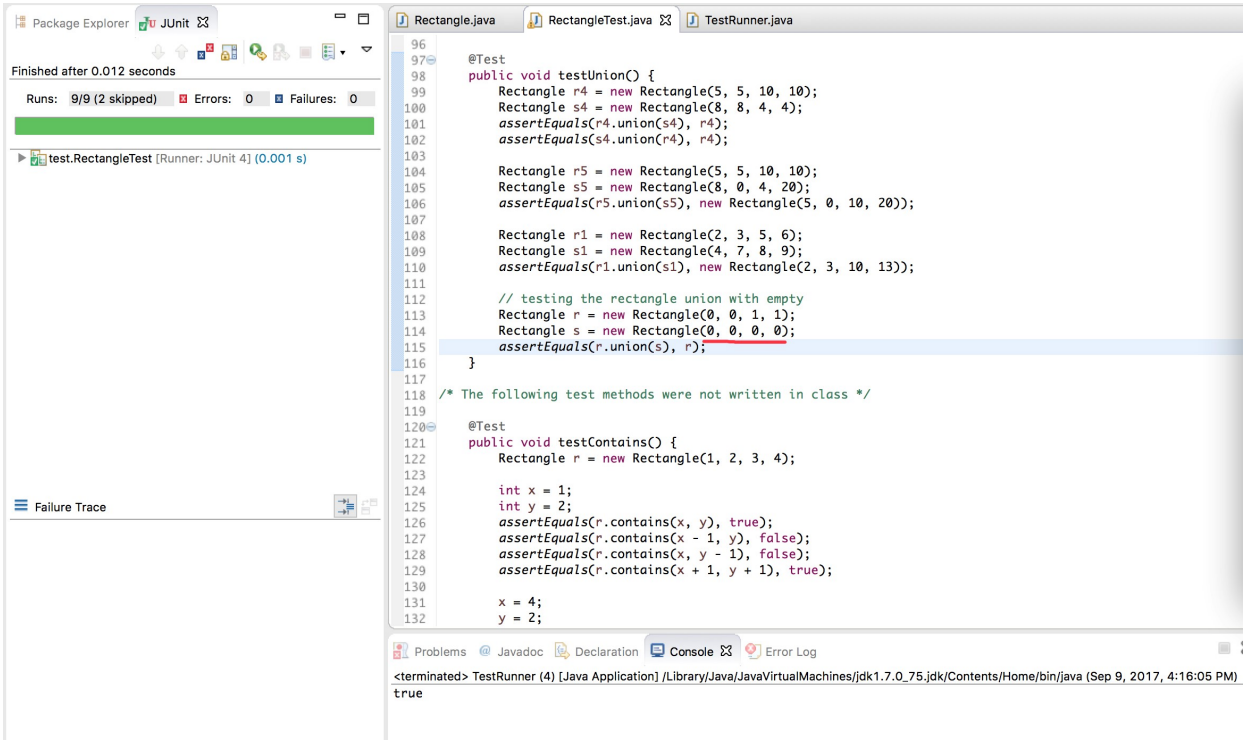


We wrote a positive test using the random generated numbers to test if the Diagonal() works, as it could cover more situations. The result passed.



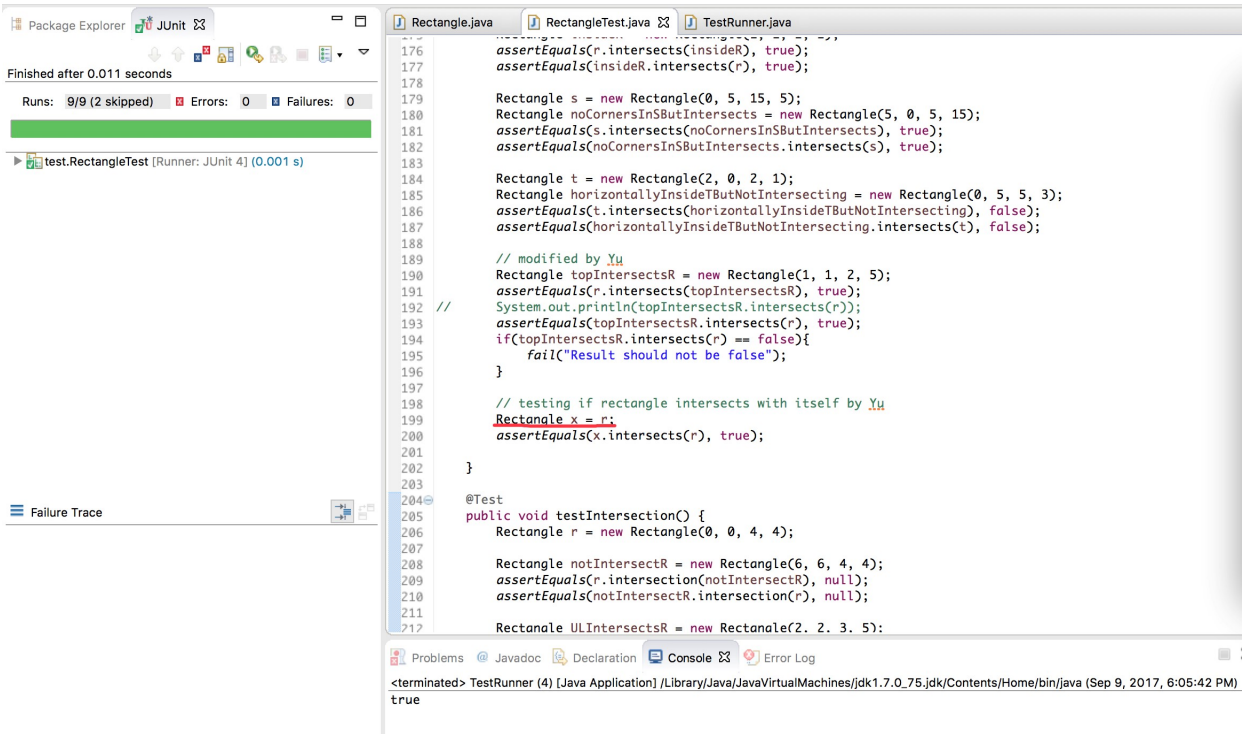
- TestUnion()

We test a special situation as the rectangle r union with an empty rectangle(0, 0, 0, 0), we expect the result to be r, and it is.



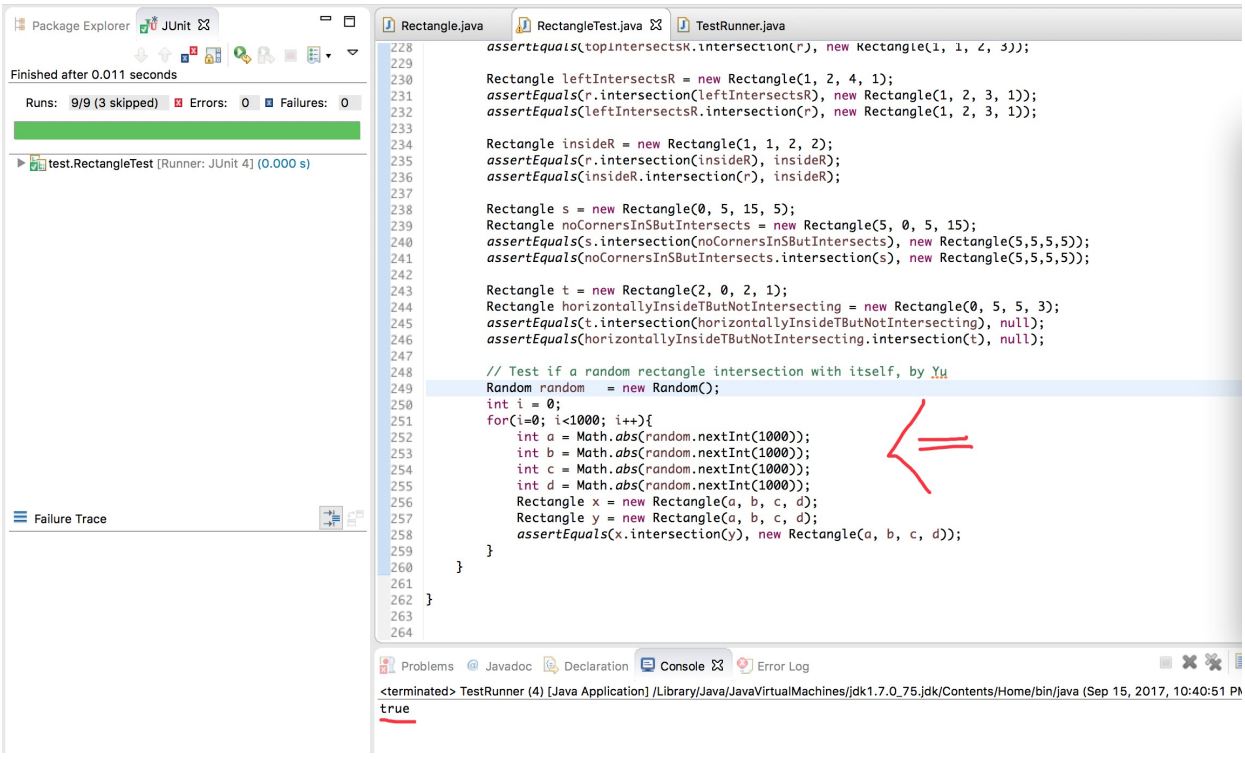
- TestIntersects()

  We test the situation when the given rectangle r intersects with a newly created rectangle of the same dimension as r, and the expected result should be r, and it is.



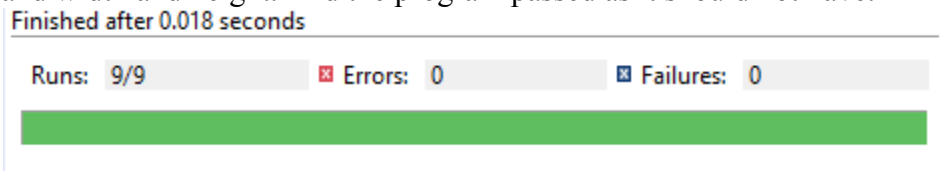
- TestIntersection()

  We test the situation when a randomly generated a rectangle intersection with itself, the expected result should still be itself, and it is. We ran the test 1000 times, they all passed.



- TestInit() & Throw an exception in Rectangle constructor:

  We wrote a negative test which contains 4 negative numbers representing the location and width and height. And the program passed as it should not have.




```
public class RectangleTest {

    // Made Rectangle throw an exception for negative parameters Mario LoPrinzi
    @Test(expected = InvalidParameterException.class)
    public void testInit() {
        Rectangle r = new Rectangle(1, 2, 3, 4);
        assertEquals(r.getX(), 1);
        assertEquals(r.getY(), 2);
        assertEquals(r.getWidth(), 3);
        assertEquals(r.getHeight(), 4);

        // testing with negative numbers, tested by Yu
        new Rectangle(-1, -1, -1, -1);
        // testing with negative width and height Mario LoPrinzi
        new Rectangle(0, 0, -1, -1);
    }
```

  We threw an exception in the constructor to catch the error. Now the error message will be shown if the invalid value was input.

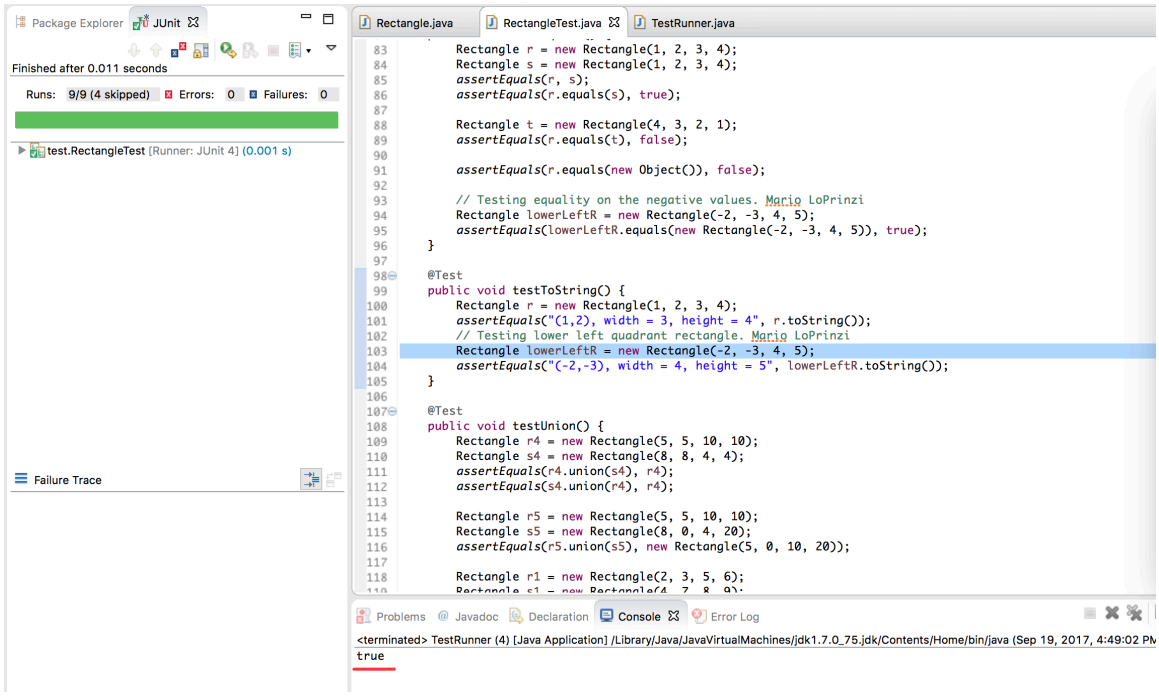
```
    // Constructor declaration
    Rectangle(int x, int y, int width, int height) {
        this.x = x;
        this.y = y;
        this.width = width;
        this.height = height;
        // Check for negative values so negative rectangles cannot be made.
        // Mario LoPrinzi
        if (this.width < 0 || this.height < 0) {
            throw new InvalidParameterException("Values must be positive.");
        }
    }
```

- TestEquals()

  We added a test for the lower left quadrant where the x and y of the Rectangle are negative.

```
    @Test
    public void testEquals() {
        Rectangle r = new Rectangle(1, 2, 3, 4);
        Rectangle s = new Rectangle(1, 2, 3, 4);
        assertEquals(r, s);
        assertEquals(r.equals(s), true);

        Rectangle t = new Rectangle(4, 3, 2, 1);
        assertEquals(r.equals(t), false);

        assertEquals(r.equals(new Object()), false);

        // Testing equality on the negative values. Mario LoPrinzi
        Rectangle lowerLeftR = new Rectangle(-2, -3, 4, 5);
        assertEquals(lowerLeftR.equals(new Rectangle(-2, -3, 4, 5)), true);
    }
```

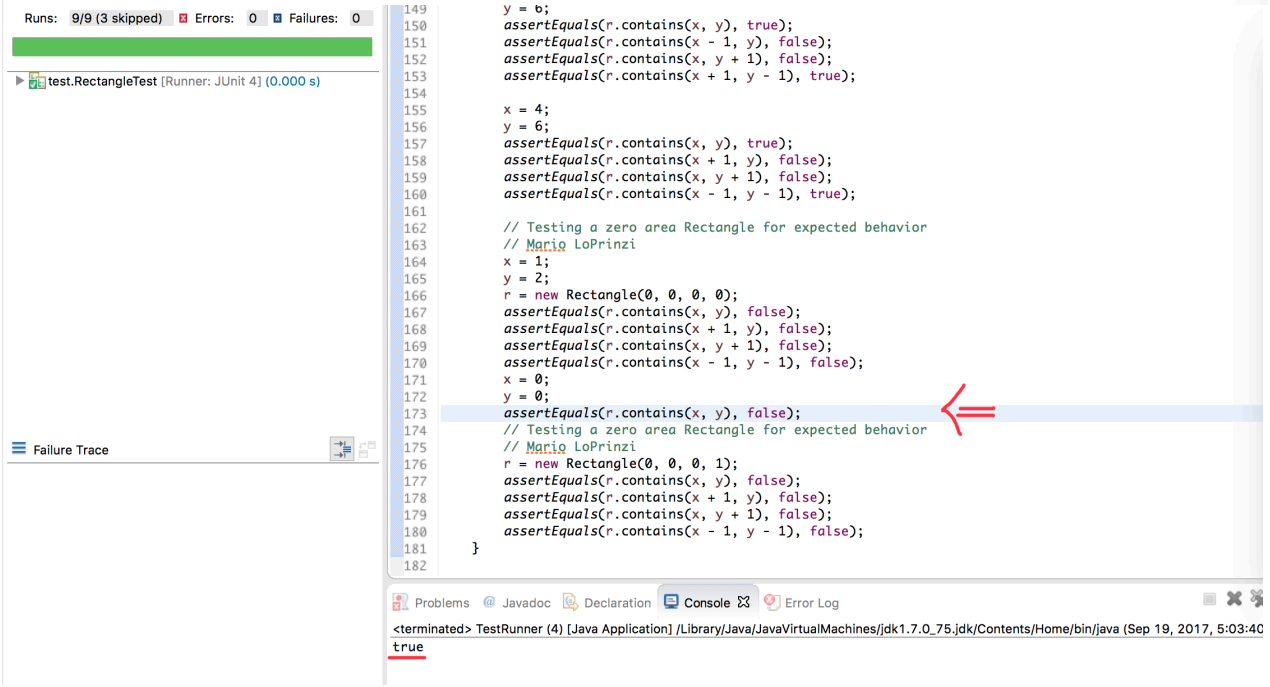
- TestToString()
  We added a negative test to make sure the test passes.




```
        Rectangle r = new Rectangle(1, 2, 3, 4);
        Rectangle s = new Rectangle(1, 2, 3, 4);
        assertEquals(r, s);
        assertEquals(r.equals(s), true);

        Rectangle t = new Rectangle(4, 3, 2, 1);
        assertEquals(r.equals(t), false);

        assertEquals(r.equals(new Object()), false);

        // Testing equality on the negative values. Mario LoPrinzi
        Rectangle lowerLeftR = new Rectangle(-2, -3, 4, 5);
        assertEquals(lowerLeftR.equals(new Rectangle(-2, -3, 4, 5)), true);
    }

    @Test
    public void testToString() {
        Rectangle r = new Rectangle(1, 2, 3, 4);
        assertEquals("(1,2), width = 3, height = 4", r.toString());
        // Testing lower left quadrant rectangle. Mario LoPrinzi
        Rectangle lowerLeftR = new Rectangle(-2, -3, 4, 5);
        assertEquals("(-2,-3), width = 4, height = 5", lowerLeftR.toString());
    }

    @Test
    public void testUnion() {
        Rectangle r4 = new Rectangle(5, 5, 10, 10);
        Rectangle s4 = new Rectangle(8, 8, 4, 4);
        assertEquals(r4.union(s4), r4);
        assertEquals(s4.union(r4), r4);

        Rectangle r5 = new Rectangle(5, 5, 10, 10);
        Rectangle s5 = new Rectangle(8, 0, 4, 20);
        assertEquals(r5.union(s5), new Rectangle(5, 0, 10, 20));

        Rectangle r1 = new Rectangle(2, 3, 5, 6);
```

- TestContains()
  We test this to make sure the zero-area rectangle can not contain a point.

```
public boolean contains(int x, int y) {
    if(this.area() !=0){
        return this.x <= x && x <= this.x + width && this.y <= y && y <= this.y + height;
    }
    else
        return false;
}
```




```
        y = 6;
        assertEquals(r.contains(x, y), true);
        assertEquals(r.contains(x - 1, y), false);
        assertEquals(r.contains(x, y + 1), false);
        assertEquals(r.contains(x + 1, y - 1), true);

        x = 4;
        y = 6;
        assertEquals(r.contains(x, y), true);
        assertEquals(r.contains(x + 1, y), false);
        assertEquals(r.contains(x, y + 1), false);
        assertEquals(r.contains(x - 1, y - 1), true);

        // Testing a zero area Rectangle for expected behavior
        // Mario LoPrinzi
        x = 1;
        y = 2;
        r = new Rectangle(0, 0, 0, 0);
        assertEquals(r.contains(x, y), false);
        assertEquals(r.contains(x + 1, y), false);
        assertEquals(r.contains(x, y + 1), false);
        assertEquals(r.contains(x - 1, y - 1), false);
        x = 0;
        y = 0;
        assertEquals(r.contains(x, y), false);
        // Testing a zero area Rectangle for expected behavior
        // Mario LoPrinzi
        r = new Rectangle(0, 0, 0, 1);
        assertEquals(r.contains(x, y), false);
        assertEquals(r.contains(x + 1, y), false);
        assertEquals(r.contains(x, y + 1), false);
        assertEquals(r.contains(x - 1, y - 1), false);
    }
```