This report contains tests that were performed to the method classify() of the Triangle class, the support source files can be found at: <https://github.com/faandree/BlackBoxTests>.

Test Report

The method that was testes was the method classify(), it has the following documentation:

public java.lang.String classify()

Classifies the triangle. Possible types, returned as a string, are:

* equilateral - if all three sides have equal length
* isosceles - if two sides have equal length
* right-angled - if one angle is a right angle
* scalene - all sides different lengths, no right angles
* impossible - if the lengths can't form a triangle

Returns:

type of the triangle as a string.

We can further notice that this method has a spelling error, instead of getting the output “isosceles” we get “isossceles”, in order for this not to interfere with the purpose of the test I considered the output “isossceles” correct (instead of “isosceles”) to validate a test where the triangle is indeed isosceles.

We can also point out that the constructor’s documentation does not provide any information about side values:

public Triangle​(int s1, int s2, int s3)

Constuctor (without error checking)

Parameters:

s1 - length of the side1 as an integer.

s2 - length of the side2 as an integer.

s3 - length of the side3 as an integer.

The same occurs in the method that can set the sides:

public [Triangle](file:///C:\Users\ASUS\Desktop\Black%20BoxTests\EX%20TRIANGLE\BlackBoxTests\dir\trianglepackage\Triangle.html) setSideLengths​(int s1, int s2, int s3)

Sets the lenghts of the sides of this triangle.

Parameters:

s1 - length of the side1

s2 - length of the side2

s3 - length of the side3

Returns:

a reference to this triangle.

Considering this we need to test integer numbers that can be outside a triangle specification, as {s1 = 0, s2 = 0, s3 = 0}, negative numbers and maximum numbers.

Parameters:

The parameters for this method are all the three sides of the triangle s1, s2, s3 even tough they are not directly provided to the method classify() they are used indirectly inside it.

-s1

-s2

-s3

Tests:

ECP#1 Passed

Parameters: 2, 2, 5

Expected Output: “isossceles”

Output: “isossceles”

@Test  
void ECP1() {  
 triangle = new Triangle(2, 2, 5);  
 *assertEquals*("isossceles", triangle.classify());  
}

ECP#2 Failed

Parameters: 4, 10, 4

Expected Output: “isossceles”

Output: “scalene”

@Test  
void ECP2() {  
 triangle = new Triangle(4, 10, 4);  
 *assertEquals*("isossceles", triangle.classify());  
}

ECP#3 Failed

Parameters: 8, 3, 3

Expected Output: “isossceles”

Output: “scalene”

@Test  
void ECP3() {  
 triangle = new Triangle(8, 3, 3);  
 *assertEquals*("isossceles", triangle.classify());  
}

ECP#4 Passed

Parameters: 10, 2, 4

Expected Output: “scalene”

Output: “scalene”

@Test  
void ECP4() {  
 triangle = new Triangle(10, 2, 4);  
 *assertEquals*("scalene", triangle.classify());  
}

ECP#6 Passed

Parameters: 4, 4, 4

Expected Output: “equilateral”

Output: “equilateral”

@Test  
void ECP6() {  
 triangle = new Triangle(4, 4, 4);  
 *assertEquals*("equilateral", triangle.classify());  
}

ECP#7 Failed

Parameters: 5, 3, 5

Expected Output: “isossceles”

Output: “scalene”

@Test  
void ECP7() {  
 triangle = new Triangle(5, 3, 5);  
 *assertEquals*("isossceles", triangle.classify());  
}

ECP#8 Passed

Parameters: 1, 5, 6

Expected Output: “scalene”

Output: “scalene”

@Test  
void ECP8() {  
 triangle = new Triangle(1, 5, 6);  
 *assertEquals*("scalene", triangle.classify());  
}

ECP#9 Passed

Parameters: 3, 3, 6

Expected Output: “isossceles”

Output: “isossceles”

@Test  
void ECP9() {  
 triangle = new Triangle(3, 3, 6);  
 *assertEquals*("isossceles", triangle.classify());  
}

ECP#10 Failed

Parameters: 5, 3, 4

Expected Output: “right-angled”

Output: “scalene”

@Test  
void ECP10() {  
 triangle = new Triangle(5, 3, 4);  
 *assertEquals*("right-angled", triangle.classify());  
}

ECP#11 Failed

Parameters: 3, 5, 4

Expected Output: “right-angled”

Output: “scalene”

@Test  
void ECP11() {  
 triangle = new Triangle(3, 5, 4);  
 *assertEquals*("right-angled", triangle.classify());  
}

ECP#12 Passed

Parameters: 3, 4, 5

Expected Output: “right-angled”

Output: “scalene”

@Test  
void ECP12() {  
 triangle = new Triangle(3, 4, 5);  
 *assertEquals*("right-angled", triangle.classify());  
}

ECP#13 Failed

Parameters: 10, 2, 2

Expected Output: “impossible”

Output: “scalene”

@Test  
void ECP13() {  
 triangle = new Triangle(10, 2, 2);  
 *assertEquals*("impossible", triangle.classify());  
}

ECP#14 Failed

Parameters: 9, 35, 15

Expected Output: “impossible”

Output: “scalene”

@Test  
void ECP14() {  
 triangle = new Triangle(9, 35, 15);  
 *assertEquals*("impossible", triangle.classify());  
}

ECP#15 Failed

Parameters: 1, 1, 3

Expected Output: “impossible”

Output: “isossceles”

@Test  
void ECP15() {  
 triangle = new Triangle(1, 1, 3);  
 *assertEquals*("impossible", triangle.classify());  
}

ECP#16 Failed

Parameters: 3, 2, 2

Expected Output: “isossceles”

Output: “scalene”

@Test  
void ECP16() {  
 triangle = new Triangle(3, 2, 2);  
 *assertEquals*("isossceles", triangle.classify());  
}

ECP#17 Passed

Parameters: 2, 2, 5

Expected Output: “isossceles”

Output: “isossceles”

@Test  
void ECP17() {  
 triangle = new Triangle(2, 2, 5);  
 *assertEquals*("isossceles", triangle.classify());  
}

ECP#19 Failed

Parameters: 5, 3, 2

Expected Output: “impossible”

Output: “scalene”

@Test  
void ECP19() {  
 triangle = new Triangle(5, 3, 2);  
 *assertEquals*("impossible", triangle.classify());  
}

ECP#20 Failed

Parameters: 4, 5, 1

Expected Output: “impossible”

Output: “scalene”

@Test  
void ECP20() {  
 triangle = new Triangle(4, 5, 1);  
 *assertEquals*("impossible", triangle.classify());  
}

ECP#21 Failed

Parameters: 2, 6, 8

Expected Output: “impossible”

Output: “scalene”

@Test  
void ECP21() {  
 triangle = new Triangle(2, 6, 8);  
 *assertEquals*("impossible", triangle.classify());  
}

BVA#1 Passed

Parameters: 1, 1, 1

Expected Output: “equilateral”

Output: “equilateral”

@Test  
void BVA1() {  
 triangle = new Triangle(1, 1, 1);  
 *assertEquals*("equilateral", triangle.classify());  
}

BVA#2 Passed

Parameters: 1, 1, 0

Expected Output: “impossible”

Output: “impossible”

@Test  
void BVA2() {  
 triangle = new Triangle(1, 1, 0);  
 *assertEquals*("impossible", triangle.classify());  
}

BVA#3 Passed

Parameters: 1, 0, 1

Expected Output: “impossible”

Output: “impossible”

@Test  
void BVA3() {  
 triangle = new Triangle(1, 0, 1);  
 *assertEquals*("impossible", triangle.classify());  
}

BVA#4 Passed

Parameters: 1, 0, 0

Expected Output: “impossible”

Output: “impossible”

@Test  
void BVA4() {  
 triangle = new Triangle(1, 0, 0);  
 *assertEquals*("impossible", triangle.classify());  
}

BVA#5 Passed

Parameters: 0, 1, 1

Expected Output: “impossible”

Output: “impossible”

@Test  
void BVA5() {  
 triangle = new Triangle(0, 1, 1);  
 *assertEquals*("impossible", triangle.classify());  
}

BVA#6 Passed

Parameters: 0, 1, 0

Expected Output: “impossible”

Output: “impossible”

@Test  
void BVA6() {  
 triangle = new Triangle(0, 1, 0);  
 *assertEquals*("impossible", triangle.classify());  
}

BVA#7 Passed

Parameters: 0, 0, 1

Expected Output: “impossible”

Output: “impossible”

@Test  
void BVA7() {  
 triangle = new Triangle(0, 0, 1);  
 *assertEquals*("impossible", triangle.classify());  
}

BVA#8 Passed

Parameters: 0, 0, 0

Expected Output: “impossible”

Output: “impossible”

@Test  
void BVA8() {  
 triangle = new Triangle(0, 0, 0);  
 *assertEquals*("impossible", triangle.classify());  
}

BVA#9 Passed

Parameters: IntMax, IntMax, IntMax

Expected Output: “impossible”

Output: “impossible”

@Test  
void BVA9() {  
 triangle = new Triangle(Integer.*MAX\_VALUE*, Integer.*MAX\_VALUE*, Integer.*MAX\_VALUE*);  
 *assertEquals*("equilateral", triangle.classify());  
}

BVA#10 Passed

Parameters: IntMax, IntMax, IntMax + 1

Expected Output: “impossible”

Output: “impossible”

@Test  
void BVA10() {  
 triangle = new Triangle(Integer.*MAX\_VALUE*, Integer.*MAX\_VALUE*, Integer.*MAX\_VALUE* + 1);  
 *assertEquals*("impossible", triangle.classify());  
}

BVA#11 Passed

Parameters: IntMax, IntMax + 1, IntMax

Expected Output: “impossible”

Output: “impossible”

@Test  
void BVA11() {  
 triangle = new Triangle(Integer.*MAX\_VALUE*, Integer.*MAX\_VALUE* + 1, Integer.*MAX\_VALUE*);  
 *assertEquals*("impossible", triangle.classify());  
}

BVA#12 Passed

Parameters: IntMax, IntMax + 1, IntMax + 1

Expected Output: “impossible”

Output: “impossible”

@Test  
void BVA12() {  
 triangle = new Triangle(Integer.*MAX\_VALUE*, Integer.*MAX\_VALUE* + 1, Integer.*MAX\_VALUE* + 1);  
 *assertEquals*("impossible", triangle.classify());  
}

BVA#13 Passed

Parameters: IntMax + 1, IntMax, IntMax

Expected Output: “impossible”

Output: “impossible”

@Test  
void BVA13() {  
 triangle = new Triangle(Integer.*MAX\_VALUE* + 1, Integer.*MAX\_VALUE*, Integer.*MAX\_VALUE*);  
 *assertEquals*("impossible", triangle.classify());  
}

BVA#14 Passed

Parameters: IntMax + 1, IntMax, IntMax + 1

Expected Output: “impossible”

Output: “impossible”

@Test  
void BVA14() {  
 triangle = new Triangle(Integer.*MAX\_VALUE* + 1, Integer.*MAX\_VALUE*, Integer.*MAX\_VALUE* + 1);  
 *assertEquals*("impossible", triangle.classify());  
}

BVA#15 Passed

Parameters: IntMax + 1, IntMax + 1, IntMax

Expected Output: “impossible”

Output: “impossible”

@Test  
void BVA15() {  
 triangle = new Triangle(Integer.*MAX\_VALUE* + 1, Integer.*MAX\_VALUE* + 1, Integer.*MAX\_VALUE*);  
 *assertEquals*("impossible", triangle.classify());  
}

BVA#16 Passed

Parameters: IntMax + 1, IntMax + 1, IntMax + 1

Expected Output: “impossible”

Output: “impossible”

@Test  
void BVA16() {  
 triangle = new Triangle(Integer.*MAX\_VALUE* + 1, Integer.*MAX\_VALUE* + 1, Integer.*MAX\_VALUE* + 1);  
 *assertEquals*("impossible", triangle.classify());  
}

BVA#17 and BVA#18 cannot be performed using IntelliJ IDEA.

BVA#19 Failed

Parameters: 10, 5, 5

Expected Output: “impossible”

Output: “scalene”

@Test  
void BVA19() {  
 triangle = new Triangle(10, 5, 5);  
 *assertEquals*("impossible", triangle.classify());  
}

BVA#20 Passed

Parameters: 10, 5, 6

Expected Output: “impossible”

Output: “scalene”

@Test  
void BVA20() {  
 triangle = new Triangle(10, 5, 6);  
 *assertEquals*("scalene", triangle.classify());  
}

BVA#21 Passed

Parameters: -1, -1, -1

Expected Output: “impossible”

Output: “impossible”

@Test  
void BVA21() {  
 triangle = new Triangle(-1, -1, -1);  
 *assertEquals*("impossible", triangle.classify());  
}

BVA#22 Passed

Parameters: 2, -1, -1

Expected Output: “impossible”

Output: “impossible”

@Test  
void BVA22() {  
 triangle = new Triangle(2, -1, -1);  
 *assertEquals*("impossible", triangle.classify());  
}

BVA#23 Passed

Parameters: -1, 2, -1

Expected Output: “impossible”

Output: “impossible”

@Test  
void BVA23() {  
 triangle = new Triangle(-1, 2, -1);  
 *assertEquals*("impossible", triangle.classify());  
}

BVA#24 Passed

Parameters: -1, -1, 2

Expected Output: “impossible”

Output: “impossible”

@Test  
void BVA24() {  
 triangle = new Triangle(-1, -1, 2);  
 *assertEquals*("impossible", triangle.classify());  
}

BVA#25 Passed

Parameters: -1, 2, 2

Expected Output: “impossible”

Output: “impossible”

@Test  
void BVA25() {  
 triangle = new Triangle(-1, 2, 2);  
 *assertEquals*("impossible", triangle.classify());  
}

BVA#26 Passed

Parameters: 2, -1, 2

Expected Output: “impossible”

Output: “impossible”

@Test  
void BVA26() {  
 triangle = new Triangle(2, -1, 2);  
 *assertEquals*("impossible", triangle.classify());  
}

BVA#27 Passed

Parameters: 2, 2,-1

Expected Output: “impossible”

Output: “impossible”

@Test  
void BVA27() {  
 triangle = new Triangle(2, 2, -1);  
 *assertEquals*("impossible", triangle.classify());  
}

Conclusions and changes:

The method handles well boundary values, the maximum and minimum ones that a triangle can have, but there are bugs recognising when a triangle is:

* Isosceles, if the equal sides are not s1 and s2.
* Right-angled, if the hypotenuse is not s3.
* Impossible, if all sides are bigger than 0 such as BVA#19 or ECP#13 .
* The method often considers a triangle as scalene when the correct designation is misjudged.
* It does a great job checking if the triangle is equilateral.

There must be a change in the way the method identifies Isosceles, Rectangles, Scalenes and Impossible Triangles.

The way that it identifies the equilateral triangle seems to be working as fine as possible, so the tests do not call for a change in this area.

The method also handles well negative numbers.

Note: Testing with IntelliJ IDEA prevents the tests to have null values or any other value that does not have an Integer representation such as strings, note that characters are valid inputs as they can be numerically represented (ASCII table).