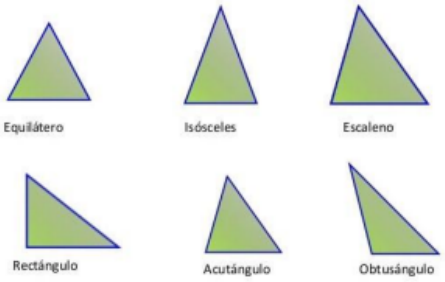
Theoretical Exercise - Problem #2

Distribution of work
Pseudocode: Team 2: Alberto, Andrés
Testing: Team 3: Marvin

0 - Problem Description

ET.02.03 Segundo Problema

Write and test a method that, accepting a triangle object (sides and angles attributes), determines the type of triangle it is. In case negative numbers or letters are passed, an exception must be thrown to indicate this situation.



1- Write, at least the pseudocode of the identified method.

```
import java.io.*;
import java.util.*;
public class Principal {
    private static Scanner sc = new Scanner(System.in);
    public static void main(String[] args) throws IOException {


        try {
            int [] sides = input_sides();
            System.out.println("Array sides: ");
            for (int i=0;i<sides.length;i++) {
                System.out.print(sides[i]+", ");
            }

            int [] angles = input_angles();

            System.out.println("Array angles: ");
            for (int i=0;i<angles.length;i++) {
                System.out.print(angles[i]+", ");
            }
            Triangle triangle = new Triangle (angles,sides);

            System.out.println ("Triangle type depending on side is: "+triangle.sideType()+", and
depending on angle is "+triangle.sideAngle());

        } catch (IOException e) {
```

```
        }
    }


    /**
     * This function takes three integer values from the user and stores them in an array, so we create
the angles of the triangle array.
     *
     * @return The method is returning an array of integers.
     */
    public static int [] input_angles(){
        System.out.println("\nIntroduce values for angles");
        int angles [] = new int [3];
        int i = 0, sum = 0, inputValue = 0;
        try{
            while(i < 2){
                System.out.println("Introduce value for position " + i);
                inputValue = introduceIntegerVariable();
                angles[i] = inputValue;
                sum += inputValue;

                if (i == 2 && sum != 180){
                    System.out.println("This is not a triangle, please try again");
                    i = 0;
                }
                i++; // Si no funciona, ponerlo antes del if.
            }

        }catch(IOException ex){

        }
        return angles;
    }
    /**
    * This function takes three integer values from the user and stores them in an array, so we create
the sides of the triangle array.
    *
    * @return The method returns an array of integers.
    */
    public static int [] input_sides() throws IOException{
        System.out.println("Introduce values of sides");
        int sides [] = new int [3];
        try {
            for (int i=0; i<3;i++) {
            System.out.println("Introduce value side number "+i);
            int number = introduceIntegerVariable();
            sides [i] = number;
        }
        } catch(InputMismatchException e) {

        }
        return sides;
    }

    public static int  introduceIntegerVariable() throws IOException {
        int integerVariable=-1;

            try {
                do {
                    System.out.println("Introduce a number bigger than 0");
                    integerVariable=sc.nextInt();
                } while (integerVariable <=0);

            } catch (InputMismatchException e) {
                System.err.println("Introduce a correct value");
                sc.next();
                introduceIntegerVariable();
            }

        return integerVariable;
    }
}
```

```
public class Triangle {
    private int[] angles;
    private int[] sides;

    // Constructor
    public Triangle(int[] angles, int[] sides) {
        this.angles = angles;
        this.sides = sides;
    }

    // Setters and Getters
    public int[] getAngles() {
        return angles;
    }

    public void setAngles(int[] angles) {
        this.angles = angles;
    }

    public int[] getSides() {
        return sides;
    }

    public void setSides(int[] sides) {
        this.sides = sides;
    }

    // Functional methods
    /**
     * It returns a string that says what type of triangle it is.
     *
     * @return The method sideType() is returning the type of triangle.
     */
     public String sideType() {
         String triangleType = "";
         if (this.sides[0] == this.sides[1] && this.sides[1] == this.sides[2]) {
             return triangleType = "It's Equilatero";
         }
         else if (this.sides[0] == this.sides[1] || this.sides[0] == this.sides[2] || this.sides[1] ==
                  this.sides[2]) {
            return triangleType = "It's Isósceles";
         }
         else if (this.sides[0] != this.sides[1] && this.sides[1] != this.sides[2] && this.sides[0] !=
                  this.sides[2]) {
            return triangleType = "It's Escaleno";
         }
         return triangleType;
     }

    /**
     * It returns the type of triangle based on the angles
     *
     * @return The method returns the type of triangle.
     */
    public String sideAngle() {
        String t_angle = "";
        for (int i = 0; i < this.angles.length; i++) {
            if (this.angles[i] > 90) {
                i = this.angles.length - 1;
                t_angle = "Osbtángulo";
            } else if (this.angles[i] == 90) {
                i = this.angles.length - 1;
                t_angle = "Rectangulo";
            } else {
                t_angle = "Acutangulo";
            }
        }
        return t_angle;
    }

}
```

2 - Identify the variables that must be considered to test the method.

The problem of determining a triangle type, judging from two aspects (“by angle” and “by sides”) presents a simple set of input values. Within this task, the authors of the code decided to use whole numbers (integers) for the problem definition.

`getAngles() : int[] -> (a1, a2, a3)`
`getSides()  : int[] -> (s1, s2, s3)`

These input variables are the main focus of the testing part and are found within the `Triangle` class. The `Principal` class realizes the functionality of a console application, does not contribute to the implemented logic to be tested, and is therefore omitted within the testing process.

Furthermore, we omit the relationships between angles and sides, as it is not asked for by the problem description to keep this example simple. For example, angles = (60, 60, 60), sides = (40, 50, 60) does not define a possible combination of angles and sides. Therefore, as seen in the upcoming chapters, the interrelationship will not be checked.

3 - Identify the test values for each one of the variables previously identified, specifying the technique used to obtain each of those values).

The determination of the test values to be used will be done using the example tables provided within the laboratory practice courses, specifying the **equivalence class**, **values** and **boundary values** for each parameter mentioned in section 2.

The test values we decide to keep are marked with a green highlighter. Since some error guessing values are ambiguous, it’s a good idea to reduce the amount of test values to the most essential ones.

Parameter	Equivalence class	Values from equivalence class	Boundary values (heavy variant)	Error guessing values

`getAngles()[i]`[1]*	`[-∞, 0]`
	`[1, 178]`
	`[179, ∞]`
		`-45`
		`60`
		`200`
			`0`
			`1, 178`
			`179`
				`-178`
				`45,50,60,7,`
				`180`

`getSides()[i]`[2]*	`[-∞, 0],` `[1, ∞]`	`-50` `40`	`0` `1`	`-100` `20000`

[1]* - Refers to each individual angle within a triangle for $i \in [0, 1, 2]$, as restrictions are identical for each one of them

[2]* - Refers to each individual side length within a triangle for $i \in [0, 1, 2]$, as restrictions are identical for each one of them

4 - Calculate the maximum possible number of test cases that could be generated from the test values.

In order to calculate the maximum number of test cases, we need to consider, that an input to this problem consists of two triplets defined as (a_1, a_2, a_3) and (s_1, s_2, s_3).

- For each angle, as defined above, we consider 12 different values:
  $(-20, 0, 1, 45, 50, 60, 70, 90, 178, 179, 180, 200)$.
- For each side defined above, we consider 5 different values:
  $(-50, 0, 1, 40, 20000)$

The amount of possible test values is determined by the number of possible sets of angles and sides: $12^3 * 5^3 = 1728 * 125 = 221184$.

5 - Define some test suites using each use.

While, in this step, we want to be using each possible variable value at least once, we also want to achieve another goal while choosing the test suits. We know of the possible outcomes of our algorithm, as well as restrictions defined for the sum of all angles. So, we want to find each-use test suits, that also…

- Form an equilateral triangle $(s_1 = s_2 \wedge s_2 = s_3)$
- Form an isosceles triangle $(s_1 = s_2 \vee s_2 = s_3 \vee s_3 = s_1)$
- Form a scalene triangle $(s_1 <> s_2 \wedge s_2 <> s_3 \wedge s_3 <> s_1)$
- Form an acute triangle $(a_1 < 90 \wedge a_2 < 90 \wedge a_3 < 90)$
- Form a right triangle $(a_1 = 90 \vee a_2 = 90 \vee a_3 = 90)$
- Form an obtuse triangle $(a_1 > 90 \vee a_2 > 90 \vee a_3 > 90)$
- Define an illegal value for the sum of all angles, so that $(a_1 + a_2 + a_3 < 180)$
- Define an illegal value for the sum of all angles, so that $(a_1 + a_2 + a_3 > 180)$
- Define an illegal value for a side, so that $(s_1 = 0 \vee s_2 = 0 \vee s_3 = 0)$
- Define an illegal value for an angle, so that $(a_1 = 0 \vee a_2 = 0 \vee a_3 = 0)$

Test suit	Variables
1	getAngles() = (60, 60, 60), getSides() = (40, 40, 40)
2	getAngles() = (60, 60, 60), getSides() = (40, 40, 1)
3	getAngles() = (60, 60, 60), getSides() = (20000, 40, 1)
4	getAngles() = (178, 1, 1), getSides() = (40, 40, 40)
5	getAngles() = (90, 45, 45), getSides() = (40, 40, 40)
6	getAngles() = (50, 60, 70), getSides() = (40, 40, 40)
7	getAngles() = (50, 50, 200), getSides() = (40, 40, 40)
8	getAngles() = (60, 60, 50), getSides() = (40, 40, 40)
9	getAngles() = (60, 60, 60), getSides() = (40, 40, 0)
10	getAngles() = (179, 1, 0), getSides() = (40, 40, 40)
11	getAngles() = (− 45, 45, 180), getSides() = (40, 40, 40)
12	getAngles() = (179, 1, 1), getSides() = (40, 40, 40)
13	getAngles() = (60, 60, 60), getSides() = (− 50, 40, 40)

6 - Define test suits to achieve pairwise coverage by using the proposed algorithm in lectures. You can check the results by means of the software PICT1

Due to the large amount of variables, the `pict` tool by Microsoft was utilized, creating a total of **163 test suits**.

164 lines (164 sloc) | 6.96 KB

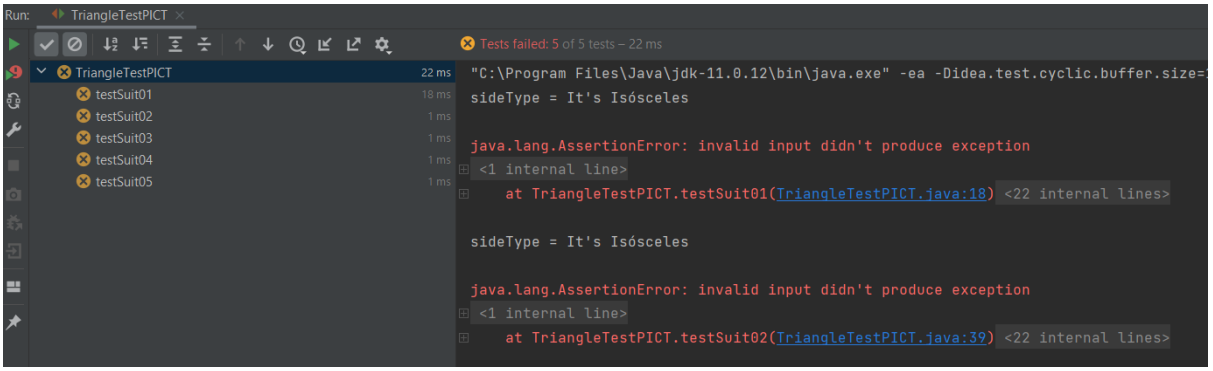
	angle_1	angle_2	angle_3	side_1	side_2	side_3
2	178	0	179	0	20000	0
3	180	179	50	-50	-50	40
4	180	45	179	40	1	1
5	0	180	-20	1	40	-50
6	70	180	180	20000	0	20000
7	-20	45	90	1	-50	20000
8	60	200	200	40	40	0
9	60	70	-20	20000	20000	40
10	200	178	178	-50	1	-50
11	60	1	90	0	0	1
12	50	60	179	-50	40	20000

The methodology of using this tool is explained within a document provided within our GitHub: [PICT_usage.md]. The results are had also been made available at: [testing_result.txt].

Using JUnit, the first five of these resulting test suits were implemented and made available at: [TriangleTestPICT.java].

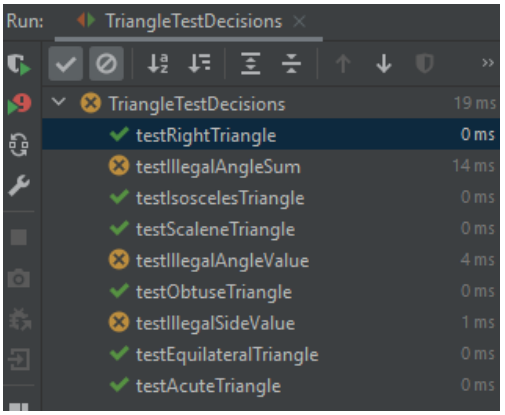
```java
@Test
public void testSuit01() throws RuntimeException {
    try {
        int[] angles = new int[]{178, 0, 179};
        int[] sides = new int[]{0, 20000, 0};

        // needs to throw exception
        Triangle t = new Triangle(angles, sides);
        String sideType = t.sideType();
    } catch (RuntimeException e) {
        return;
    }

    // exception was not thrown
    org.junit.Assert.fail("invalid input didn't produce exception");
}
```

Running these tests, we can read the results from the console. In this case, the code does not pass the first five vectors. This implies that the underlying code does not check for invalid inputs. Using a change request, we'd use this result to invoke a fix.



7 - For code snippets that include decisions, propose a set of test cases to achieve coverage of decisions.

To create the requested set of test cases, we can use the list defined in section 5 one to one. The decisions made within the code consist of the different types of triangles we want to detect - Therefore, we need to choose test cases with values for each detectable triangle type. We also need to check for possible illegal values, as this is also a type of decision (throw an expectation or don't) the code should be able to make. After defining these tests, details provided at TriangleTestDecisions.java, we see that six out of nine tests are passed.



We observe, that the triangle types are correctly determined given the initial restraints. Also, as we found out in section 7, illegal parameter combinations are not detected and falsely produce a result type, when an error should be produced instead.

Looking at the code coverage of 80%, we see that the getters and setters are not used by the internal logic. Also, we can see that the variable `triangleType` in `sideType()` is unused and one `return` statement always remains unused. The important decisions, however, are covered. The full report is available at: DecisionCoverageReport.pdf.

8 - For code snippets that include decisions, propose test case sets to achieve MC/DC coverage.

In this test, we need to produce all different outcomes (triangle types) using all logically possible different combination. This test will first define the different logic checks in a table, looking at the methods sideType() and sideAngle() respectively, and then define tests in JUnit. Since we already know of the absence of checks for illegal values, we want to test the logic for determining all six different triangle types only.

New test cases (in comparison to section 7) have been highlighted.

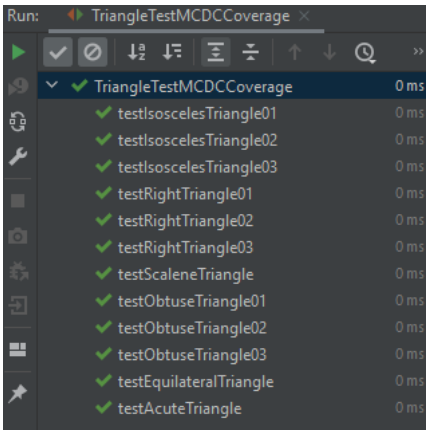
sideType()

A (side_1)	B (side_1)	C (side_1)	Logic	Dominant Condition	Expected Output value
60	60	60	A = B = C	A, B, C	"It's Equilatero"
40	40	1	(A=B) ∨ (B=C) ∨ (C=A)	A,B	“It's Isósceles”
1	40	40	(A=B) ∨ (B=C) ∨ (C=A)	B,C	“It's Isósceles”
40	1	40	(A=B) ∨ (B=C) ∨ (C=A)	A,C	“It's Isósceles”
50	70	60	A != B != C	A,B,C	"It's Escaleno"

sideAngle()

A (angle_1)	B (angle_2)	C (angle_3)	Logic	Dominant Condition	Expected Output value
50	60	70	(A<90) ∧ (B<90) ∧ (C<90)	A, B, C	“Acutangulo“
90	45	45	(A=90) ∨ (B=90) ∨ (C=90)	A	“Rectangulo“
45	90	45	(A=90) ∨ (B=90) ∨ (C=90)	B	“Rectangulo“
45	45	90	(A=90) ∨ (B=90) ∨ (C=90)	C	“Rectangulo“
178	1	1	(A>90) ∨ (B>90) ∨ (C>90)	A	“Osbtángulo“
1	178	1	(A>90) ∨ (B>90) ∨ (C>90)	B	“Osbtángulo“
1	1	178	(A>90) ∨ (B>90) ∨ (C>90)	C	“Osbtángulo“

Running the newly defined test cases, available at [TriangleTestMCDCCoverage.java], we see that all test cases pass.



9 - Comment on the results of the number of test cases obtained in section 4, 5, and 6, as well as the execution of the oracles: what could be said about the coverage achieved?

As discussed in section 9, the coverage for the available code is not optimal, as we can see from the statistics produced:

Coverage Summary for Class: Triangle (<empty package name>)

Class	Class, %	Method, %	Line, %
Triangle	100% (1/1)	42,9% (3/7)	80% (20/25)

The bad method coverage can be fixed by using the defined getters and setters instead of `this` statements.

```
11      // Setters and Getters
12      public int[] getAngles() {
13          return angles;
14      }
15
16      public void setAngles(int[] angles) {
17          this.angles = angles;
18      }
19
20      public int[] getSides() {
21          return sides;
22      }
23
24      public void setSides(int[] sides) {
25          this.sides = sides;
26      }
```

Furthermore, the missing 20% of line coverage may be achieved by simple refactoring of the code of `sideType()`.

```
35      public String sideType() {
36          String triangleType = "";
37          if (this.sides[0] == this.sides[1] && this.sides[1] == this.sides[2]) {
38              return triangleType = "It's Equilatero";
39          } else if (this.sides[0] == this.sides[1] || this.sides[0] == this.sides[2] || this.sides[1] == this.sides[2]) {
40              return triangleType = "It's Isósceles";
41          } else if (this.sides[0] != this.sides[1] && this.sides[1] != this.sides[2] && this.sides[0] != this.sides[2]) {
42              return triangleType = "It's Escaleno";
43          }
44          return triangleType;
45      }
```