Boundary Value Analysis (BVA) using Junit

Boundary Value Analysis is one kind of specification based testing. In this kind of testing, we need to create test cases based on the software specification, not based on the coding structure. BVA based test cases are created based on the value of the ***independent*** and ***physical quantities***. To do BVA using JUnit5, it will be better to follow Parameterized test in Junit.

In this tutorial, we will see how to do Normal Boundary Value Analysis and Robust Boundary Value Analysis using JUnit.

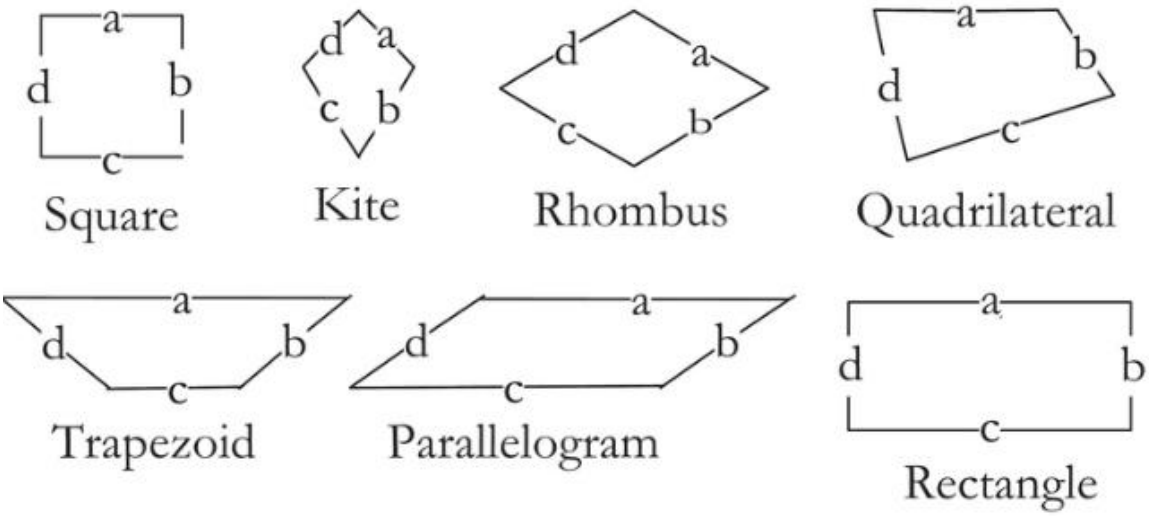
Problem Specification:

The Quadrilateral Program It accepts four integers, side1, side2, side3 and side4, as input. These are taken to be sides of a four-sided figure and they must satisfy the following conditions:

1. c1. $1 \leq a \leq 200$ (top)
2. c2. $1 \leq b \leq 200$ (left side)
3. c3. $1 \leq c \leq 200$ (bottom)
4. c4. $1 \leq d \leq 200$ (right side)

The output of the program is the type of quadrilateral determined by the four sides: Square, Rectangle, Trapezoid, Kite or General. (Since the problem statement only has information about lengths of the four sides, a square cannot be distinguished from a rhombus, similarly, a parallelogram cannot be distinguished from a rectangle.)

1. A *square* has two pairs of parallel sides (a||c, b||d), and all sides are equal (a = b = c = d).
2. A *kite* has two pairs of equal sides, but no parallel sides (a = d, b = c).
3. A *trapezoid* has one pair of parallel sides (a||c) and one pair of equal sides (b = d).
4. A *rectangle* has two pairs of parallel sides (a||c, b||d) and two pairs of equal sides (a = c, b = d).
5. A *scalene quadrilateral* has four sides, none equal and none parallels (aka a trapezium).
6. Other options can create a *General* quadrilateral.



Solution

We will create test cases by following both Normal Boundary Value Testing and Robust Boundary Value Testing.

Normal Boundary Value Analysis:

Test cases for 1 input variable:

Test cases will follow 5 input variable values: minimum value (**min**), just above the minimum (**min+1**), a nominal value (**nom**), just below their maximum (**max-1**), and maximum value (**max**).

***Test cases for more than 1 input variable*:**

Test cases will be created by holding the value of all variables except one at their nominal values, and allowing the one variable assume 5 input variable values : minimum value (**min**), just above the minimum (**min+1**), a nominal value (**nom**), just below their maximum (**max-1**), and maximum value (**max**).

Robust Boundary Value Analysis:

Test cases for 1 input variable:

Test cases will follow 7 input variable values: just below the minimum (**min-1**), minimum value (**min**), just above the minimum (**min+1**), a nominal value (**nom**), just below their maximum (**max-1**), and maximum value (**max**), just above the maximum (**max+1**).

***Test cases for more than 1 input variable*:**

Test cases will be created by holding the value of all variables except one at their nominal values, and allowing the one variable assume 7 input variable values : just bellow the minimum (**min-1**), minimum value (**min**), just above the minimum (**min+1**), a nominal value (**nom**), just below their maximum (**max-1**), and maximum value (**max**), just above the maximum (**max+1**).

Normal Boundary Value Analysis for Quadrilateral Problem

```
@DisplayName("Normal Boundary Value Test")
@ParameterizedTest(name="topSide= {0},bottomSide= {1},leftSide= {2},"
        + "rightSide={3},ExpectedResult= {4}")
@CsvSource({"100,100,100,1,General","100,100,100,2,General","100,100,100,100,Square"
    ,"100,100,100,199,General","100,100,100,200,General",
    "100,100,1,100,General","100,100,2,100,General",
    "100,100,199,100,General","100,100,200,100,General",
    "100,1,100,100,Trapezoid","100,2,100,100,Trapezoid",
    "100,199,100,100,Trapezoid","100,200,100,100,Trapezoid",
    "1,100,100,100,Trapezoid","2,100,100,100,Trapezoid",
    "199,100,100,100,Trapezoid","200,100,100,100,Trapezoid"})
void testQuadrilateral1(int topSide, int bottomSide,int leftSide,int rightSide,String result) {
    Q.setSide(topSide, bottomSide, leftSide, rightSide);
    assertEquals(result,Q.classify());
}
```

Robust Boundary Value Analysis for Quadrilateral Problem

```
@DisplayName("Robust Boundary Value Test")
@ParameterizedTest(name="topSide= {0},bottomSide= {1},leftSide= {2},"
        + "rightSide={3},ExpectedResult= {4}")
@CsvSource({"100,100,100,1,General","100,100,100,0,OUT_OF_RANGE","100,100,100,2,General",
    "100,100,100,100,Square","100,100,100,199,General","100,100,100,200,General",
    "100,100,100,201,OUT_OF_RANGE",
    "100,100,0,100,OUT_OF_RANGE","100,100,1,100,General","100,100,2,100,General",
    "100,100,199,100,General","100,100,200,100,General","100,100,201,100,OUT_OF_RANGE",
    "100,0,100,100,OUT_OF_RANGE","100,1,100,100,Trapezoid","100,2,100,100,Trapezoid",
    "100,199,100,100,Trapezoid","100,200,100,100,Trapezoid","100,201,100,100,OUT_OF_RANGE",
    "0,100,100,100,OUT_OF_RANGE","1,100,100,100,Trapezoid","2,100,100,100,Trapezoid",
    "199,100,100,100,Trapezoid","200,100,100,100,Trapezoid","201,100,100,100,OUT_OF_RANGE"})
void testQuadrilateral2(int topSide, int bottomSide,int leftSide,int rightSide,String result) {
    Q.setSide(topSide, bottomSide, leftSide, rightSide);
    assertEquals(result,Q.classify());
}
```