# CONTINUOUS ASSESMENT LABORATORY 2

## Software Engineering

Universidad de Alcalá

MAYO DE 2023

DIEGO DÍAZ VIDAL
MARCO FERNÁNDEZ PÉREZ
PABLO ALONSO ZAPATERO
Grado en Ingeniería Informática

# CONTENTS

# 1. The Project

For this Project, we had to choose a mathematical function already implemented in Java (or implement it ourselves) to carry out several tasks upon it.
We chose to implement not one, but two mathematical functions, those being:

- Extended Euclidean algorithm:
  This well know algorithm, created by the mathematician Euclid, is vastly used in arithmetic and computer programming. It is an extension to the Euclidean algorithm.
  Its main goal is to calculate the greatest common divisor (gcd) of integers *a* and *b*.

- 3x3 matrix determinant calculator:
  Equally loved and feared by 2 BACH students, Cramer´s rule is the solution to calculating the determinant of a 3x3 matrix.

We chose to implement these two mathematical functions because of their utility and versatility.

# 2. The Implementation

Since we had not one, but two functions to implement, we couldn´t just create a single class with a main method, so we came up with the idea of creating a class named "Calculator" with two methods defined, one for each function.

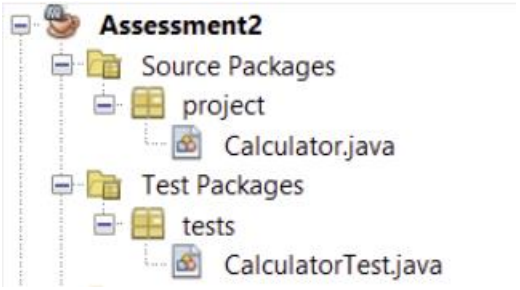
The syntax is as follows:

```
package project;

/**
 *
 * @authors Diego Díaz, Pablo Alonso, Marco Fernández
 */
public class Calculator {

    // Method 1 of class Calculator
    public static int extendedEuclidean(int a, int b) {
        if (b == 0) {
            return a;
        } else {
            return extendedEuclidean(b, a % b);
        }
    }

    // Method 2 of class Calculator
    public static double determinant3Matrix(double[][] matrix) {
```

```
        double det = 0;
        det += matrix[0][0] * matrix[1][1] * matrix[2][2];
        det += matrix[0][1] * matrix[1][2] * matrix[2][0];
        det += matrix[0][2] * matrix[1][0] * matrix[2][1];
        det -= matrix[0][2] * matrix[1][1] * matrix[2][0];
        det -= matrix[0][1] * matrix[1][0] * matrix[2][2];
        det -= matrix[0][0] * matrix[1][2] * matrix[2][1];
        return det;
    }
}
```

# 3. The Testing

For testing these two functions, we had to create a separate class in the "Test Packages" package, in which we would implement our various tests.




## 3.1. JUnit and mock tests

In order to creat the necessary **JUnit** and **mock** tests to cover for all the funcionality, we had to import the following libraries:

```
// Imports
import org.junit.After;
import org.junit.AfterClass;
import org.junit.Before;
import org.junit.BeforeClass;
import org.junit.Test;
import static org.junit.Assert.*;
import org.junit.jupiter.api.Assertions;
import org.junit.jupiter.api.extension.ExtendWith;
import org.mockito.Mock;
import org.mockito.junit.jupiter.MockitoExtension;
import project.Calculator;
```

The next step was to implement the tests.

For this purpose, we declared two mocks to test both functions:

```
@Mock
private Calculator euclideanObject;
private Calculator matrixObject;
```

Finally, we created the tests:

- Tests for the **extendedEuclidean** method:

```
/**
* Test 1 of extendedEuclidean method, of class Calculator.
*/
@Test
public void testExtendedEuclidean1() {

    System.out.println("extendedEuclidean - testing method - test 1");

    int a = 348;
    int b = 228;
    int expResult = 12;

    int result = euclideanObject.extendedEuclidean(a, b);

    assertEquals(expResult, result, 0);
}

/**
* Test 2 of extendedEuclidean method, of class Calculator.
*/
@Test
public void testExtendedEuclidean2() {

    System.out.println("extendedEuclidean - testing method - test 2");

    int a = 234;
    int b = 69;
    int expResult = 3;

    int result = euclideanObject.extendedEuclidean(a, b);

    assertEquals(expResult, result, 0);
}

/**
* Test 3 of extendedEuclidean method, of class Calculator.
*/
@Test
public void testExtendedEuclideanWrongParameters() {

    System.out.println("extendedEuclidean - wrong parameters test");

    try {
        euclideanObject.extendedEuclidean(12, 42);
    } catch (Exception e) {
        assertEquals(e.getMessage(), "Second argument must be"
                + " equal or greater than the first argument.");
    }
```

```
}

/**
* Test 4 of extendedEuclidean method, of class Calculator.
*/
@Test
public void testExtendedEuclideanEqualParameters() {

    System.out.println("extendedEuclidean - equal parameters test");

    int a = 64;
    int b = 64;
    int expResult = 64;

    int result = euclideanObject.extendedEuclidean(a, b);

    assertEquals(expResult, result, 0);
}
```

- Tests for the **determinant3Matrix** method:

```
/**
* Test 1 of determinant3Matrix method, of class Calculator.
*/
@Test
public void testDeterminantMatrix1() {

    System.out.println("determinantMatrix - testing method - test 1");

    double[][] matrix = {{2, 3, 1}, {4, -1, 0}, {2, 1, 2}};

    assertEquals(matrixObject.determinant3Matrix(matrix), -22, 0);
}

/**
* Test 2 of determinant3Matrix method, of class Calculator.
*/
@Test
public void testDeterminantMatrix2() {

    System.out.println("determinantMatrix - testing method - test 2");

    double[][] matrix = {{2, 2, 2}, {4, 4, 4}, {-5, 11, 32}};

    assertEquals(matrixObject.determinant3Matrix(matrix), 0, 0);
}

/**
* Test 3 of determinant3Matrix method, of class Calculator.
*/
@Test
public void testDeterminantMatrix3() {

    System.out.println("determinantMatrix - testing method - test 3");

    double[][] matrix = {{3, 4, 2}, {5, 0, 1}, {3, 2, 3}};

    assertEquals(matrixObject.determinant3Matrix(matrix), -34, 0);
```

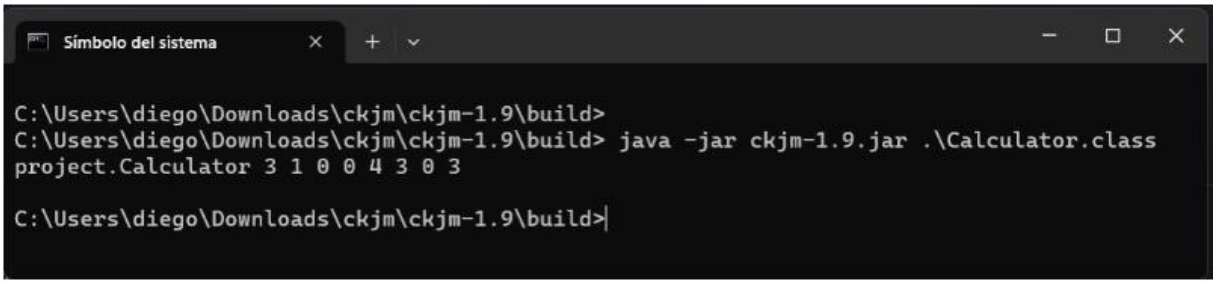
```
}
```

- Last, but not least, we created and checked for some **Java Exceptions**:

```
/**
* Test 5 of extendedEuclidean method, of class Calculator.
*/
@Test
public void testExtendedEuclideanExpectedException() {
    // Testing division by 0 exception
    System.out.println("extendedEuclidean - testing division by 0 exception");

    Assertions.assertThrows(ArithmeticException.class,
            () -> euclideanObject.extendedEuclidean(24, 8/0));
}

/**
* Test 4 of determinant3Matrix method, of class Calculator.
*/
@Test
public void testDeterminant3MatrixExpectedException() {
    // Testing method with 2x2 matrix
    System.out.println("determinantMatrix - testing invalid matrix
exception");

    double[][] matrix = {{1, 2}, {3, 4}};
    assertThrows(ArrayIndexOutOfBoundsException.class,
            () -> matrixObject.determinant3Matrix(matrix));
}
```

# 4. Calculating Chidamber and Kemerer Java Metrics

For this, we executed the following command:




Which processes the bytecode of the compiled Java files and calculates for each class (for the class “Calculator”, in this case) the following metrics:

- **WMC**: Weighted methods per class: **score of 3**
  It is the sum of the complexities of its methods. The ckjm program assigns

a complexity value of 1 to each method, and therefore the value of the WMC is equal to the number of methods in the class.

- **DIT**: Depth of Inheritance Tree: **score of 1**
  Provides a measure of the inheritance levels from the object hierarchy top. In Java, since all classes inherit "Object", the minimum value is 1.

- **NOC**: Number of Children: **score of 0**
  It measures the number of immediate descendants of the class.

- **CBO**: Coupling between object classes: **score of 0**
  This metric represents the number of classes coupled to a given class. This coupling can occur through method calls, field accesses, inheritance, arguments, return types, and exceptions.

- **RFC**: Response for a Class: **score of 4**
  Measures the number of different methods that can be executed when an object of this class receives a message.

- **LCOM**: Lack of cohesion in methods: **score of 3**
  This metric counts the sets of methods in a class that are not related through the sharing of some of the class's fields.

- **Ca**: Afferent coupling: **score of 0**
  It is a measure of how many other classes use the specific class.

- **NPM**: Number of Public Methods for a class: **score of 3**
  This metric simply counts all the methods in a class that are declared as public. It can be used to measure the size of an API provided by a package.