Proba de Desenvolvemento de Interfaces: UD8

| 8.1 - CA3.1: 20% (ME) | 8.2 - CA3.2: 20% (ME) |
|---|---|

**Apellidos, Nombre: Iglesias Nieto, Rodrigo**

**Fecha: 06/03/24**

## Unidad 8: Realización de pruebas

***CA2.1 - Empaquetáronse os compoñentes que require a aplicación. (20% - ME)***

**8.1 Dado el siguiente código en Java, indica qué estrategia de pruebas seguirías para minimizar la posibilidad de existencia de errores:**

```
package calculator;

public class Calculator {
    public double divide(double operand1, double operand2) {
        return operand1 / operand2;
    }
}
```

Primero generaríamos unas pruebas utilizando los valores máximos y mínimos que soportan las variables de tipo Double en este caso. Para esto podemos utilizar las variables de la clase Double MAX_VALUE y MIN_VALUE. Debemos hacer una prueba dividiendo máximo entre máximo y mínimo entre mínimo para comprobar que no tenemos ningún desborde de memoria de las variables. Un código de ejemplo para estas dos pruebas sería el siguiente:

```
@Test
public void testDivide1() {
    System.out.println(x: "divide");
    double operand1 = Double.MIN_VALUE;
    double operand2 = Double.MIN_VALUE;
    DecimalCalculator instance = new DecimalCalculator();
    double expResult = 1;
    double result = instance.divide(operand1, operand2);
    assertEquals(expected:expResult, actual: result, delta: 0);
}
```

```
@Test
public void testDivide2() {
    System.out.println(x: "divide");
    double operand1 = Double.MAX_VALUE;
    double operand2 = Double.MAX_VALUE;
    DecimalCalculator instance = new DecimalCalculator();
    double expResult = 1;
    double result = instance.divide(operand1, operand2);
    assertEquals(expected:expResult, actual: result, delta: 0);
}
```

En este ejemplo de pruebas utilizamos un delta de 0 y el resultado de ambos casos es 1

Podemos probar con otros dos valores cualquieras que estén entre el máximo y mínimo para comprobar que da correctamente el resultado

```
@Test
public void testDivide3() {
    System.out.println(x: "divide");
    double operand1 = 10.0;
    double operand2 = 2.0;
    DecimalCalculator instance = new DecimalCalculator();
    double expResult = 5.0;
    double result = instance.divide(operand1, operand2);
    assertEquals(expected: expResult, actual: result, delta: 0);
}
```

En este caso utilizamos los valores 10 como numerador y 2 como denominador. El resultado será 5.0 y el delta usado es 0.

En el caso de que nuestro método sea usado muchas veces por segundo podemos probar a usarlo en un bucle con muchos usos para calcular cuanto tiempo tardaría en realizar por ejemplo 1000 llamadas al método. En el caso de hacer esta prueba daría igual que números dividir ya que siempre tardará lo mismo independientemente del número, aún así, debemos tener en cuenta el tiempo que se pierde calculando la hora del sistema.

Otra posible prueba sería pasando como denominador un número 0, en esta prueba buscaríamos la excepción *java.lang.ArithmeticException*.

***CA3.2 - Realizáronse probas de integración dos elementos. (20% - ME)***

**8.2 Suponiendo que acabas de implementar la clase Service que gestiona los datos de una reparación y que te encuentras implementando la clase Computer que hace uso de la clase Service, indica la estrategia de pruebas que seguirías para minimizar la posibilidad de error en la integración de las dos clases.**

Siempre es mejor implementar una clase con otra cuando está comprobado su funcionamiento, por lo que primero debemos comprobar la clase Service en este caso y al implementarla con la clase Computer realizar las pruebas de esta clase.

De esta manera minimizaremos la posibilidad de fallos con la clase Service y nos centraremos en el funcionamiento de la clase Computer.

Algunos ejemplos de pruebas para estas clases serían crear objetos y utilizar todos los métodos: getters, setters, toString() y constructores. Para poder verificar la correcta creación de objetos de esta clase.

Una vez comprobado que se pueden generar objetos correctamente podremos probar con listas de estos objetos añadiendo, borrando o editando instancias de estas clases.