Proba de Desenvolvemento de Interfaces: UD8

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

Apellidos, Nombre: Maceira Barca, Xian

Fecha: 06/03/2024

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;
    }
}
```

Las primeras pruebas que deberíamos realizar son las pruebas unitarias utilizando los valores máximos y mínimos de las variables. Para ello podemos asignar en este caso a las variables los valores **Double.MAX_VALUE y Double.MIN_VALUE.**

Debemos dividir el valor máximo entre el máximo y el mínimo entre el mínimo, usando el valor 1.0 como resultado esperado y un delta de 0 se deberían cumplir las pruebas.

```
@Test
public void testDivide1() {
    System.out.println(x: "divide1");
    double operand1 = Double.MIN_VALUE;
    double operand2 = Double.MIN_VALUE;
    DecimalCalculator instance = new DecimalCalculator();
    double expResult = 1.0;
    double result = instance.divide(operand2, operand1);
    assertEquals(expected:expResult, actual: result, delta: 0);

}
```

```
@Test
public void testDivide2() {
    System.out.println(x: "divide2");
    double operand1 = Double.MAX_VALUE;
    double operand2 = Double.MAX_VALUE;
    DecimalCalculator instance = new DecimalCalculator();
    double expResult = 1.0;
    double result = instance.divide(operand2, operand1);
    assertEquals(expected:expResult, actual: result, delta: 0);

}
```

Otra de las pruebas que se pueden realizar es la prueba de regresión que consiste en intercambiar la posición del operando1 con el operando2, para esta prueba asignamos los valores 10.0 y 2.0 a las variables y ponemos de resultado esperado a 5.0 con un delta de 0, con esta prueba, si alguien modifica el código e intercambia los operando de posición ya tenemos un test para cubrir el error.

```
@Test
public void testDivide3() {
    System.out.println(x: "divide3");
    double operand1 = 10.0;
    double operand2 = 2.0;
    DecimalCalculator instance = new DecimalCalculator();
    double expResult = 5.0;
    double result = instance.divide(operand2:operand1, operand1:operand2);
    assertEquals(expected:expResult, actual: result, delta: 0);

}
```

Si nuestra aplicación esta pensada para ser usada por una gran cantidad de usuarios también sería necesario realizarle pruebas de volumen y estrés, de esta forma nos aseguraríamos de que la aplicación no falle cuando se le soliciten múltiples tareas.

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.

Para implementar la clase Computer de forma segura, lo más recomendable es realizar con anterioridad las pruebas unitarias necesarias para la clase Service y así asegurarnos de que tenemos cubiertos los posibles errores que nos proporcione la clase

Una vez la clase Service funciona correctamente, la óptimo es probar el funcionamiento de la clase Computer, lo cual se puede lograr comprobando todos lo métodos de la clase (getters, setters, e.t.c.) y generando objetos de esa clase. Si todo funciona de la manera esperada ya podemos continuar con la integración de las clases.