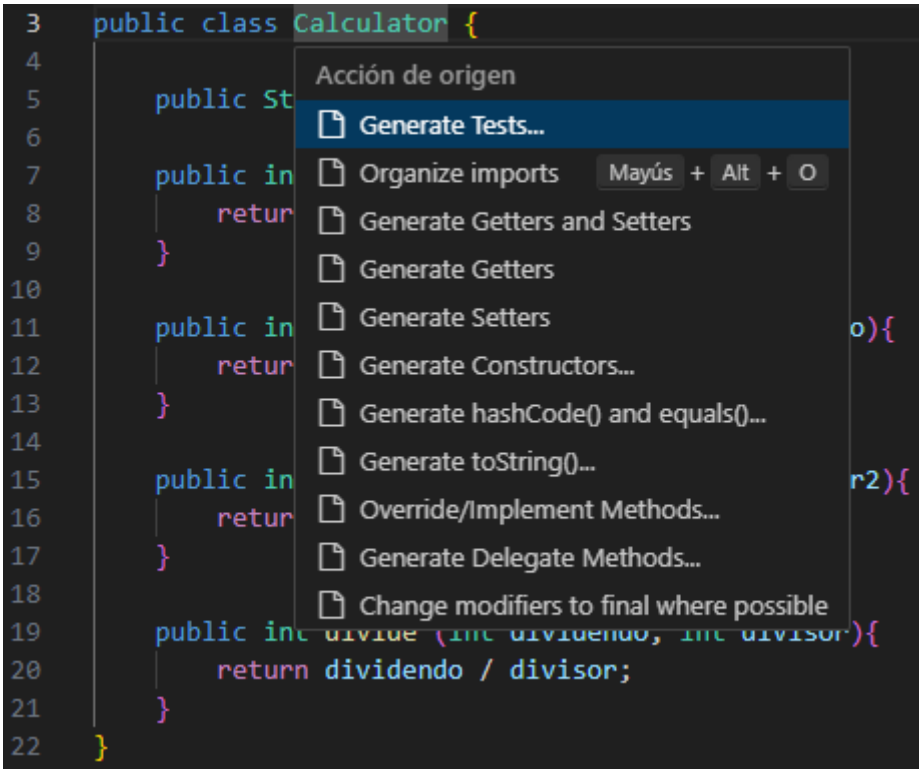
Ejercicio 1: Calculadora Simple

Crea una clase Calculator que tenga métodos para sumar, restar, multiplicar y dividir dos números. Luego, escribe pruebas unitarias para cada uno de estos métodos para verificar su funcionamiento.

En primer lugar, creo la clase “Calculator”, con todas sus funciones:

```
public class Calculator {

    public String nombre = "";

    public int suma(int sumando1, int sumando2){
        return sumando1 + sumando2;
    }

    public int resta (int minuendo, int sustraendo){
        return minuendo - sustraendo;
    }

    public int multiplica (int factor1, int factor2){
        return factor1 * factor2;
    }

    public int divide (int dividendo, int divisor){
        return dividendo / divisor;
    }
}
```

Ahora, hacemos click derecho sobre la clase, pulsamos “Acción de código fuente” > “Generar test”.



Una vez hayamos generado el archivo para test, haremos las funciones que realizarán los test.

```
public void testSuma_2_y_3() {
        //Arrange
        int sumando1 = 2;
        int sumando2 = 3;
        int sumaTotal = 5;
        Calculator calculator = new Calculator();

        //Act
        int resultadoReal = calculator.suma(sumando1, sumando2);

        //Assert
        Assert.assertEquals(sumaTotal, resultadoReal);
    }
```

Ahora veremos que significan esté código por partes.

En la parte de “Arrange” organizaremos lo que vamos a hacer. Declaramos variables y creamos instancias.

```
//Arrange
        int sumando1 = 2;
        int sumando2 = 3;
        int sumaTotal = 5;
        Calculator calculator = new Calculator();
```

En la parte de “Act” tomaremos las acciones necesarias para tomar hacer el test adecuadamente.

```
        //Act
        int resultadoReal = calculator.suma(sumando1, sumando2);
```

Por último, en la parte de “Assert”, verificaremos los resultados.

```
       //Assert
        Assert.assertEquals(sumaTotal, resultadoReal);
```

Para acabar, podemos realizar al menos un par de pruebas por cada función con distintos valores para tener mayor certeza de un buen funcionamiento.

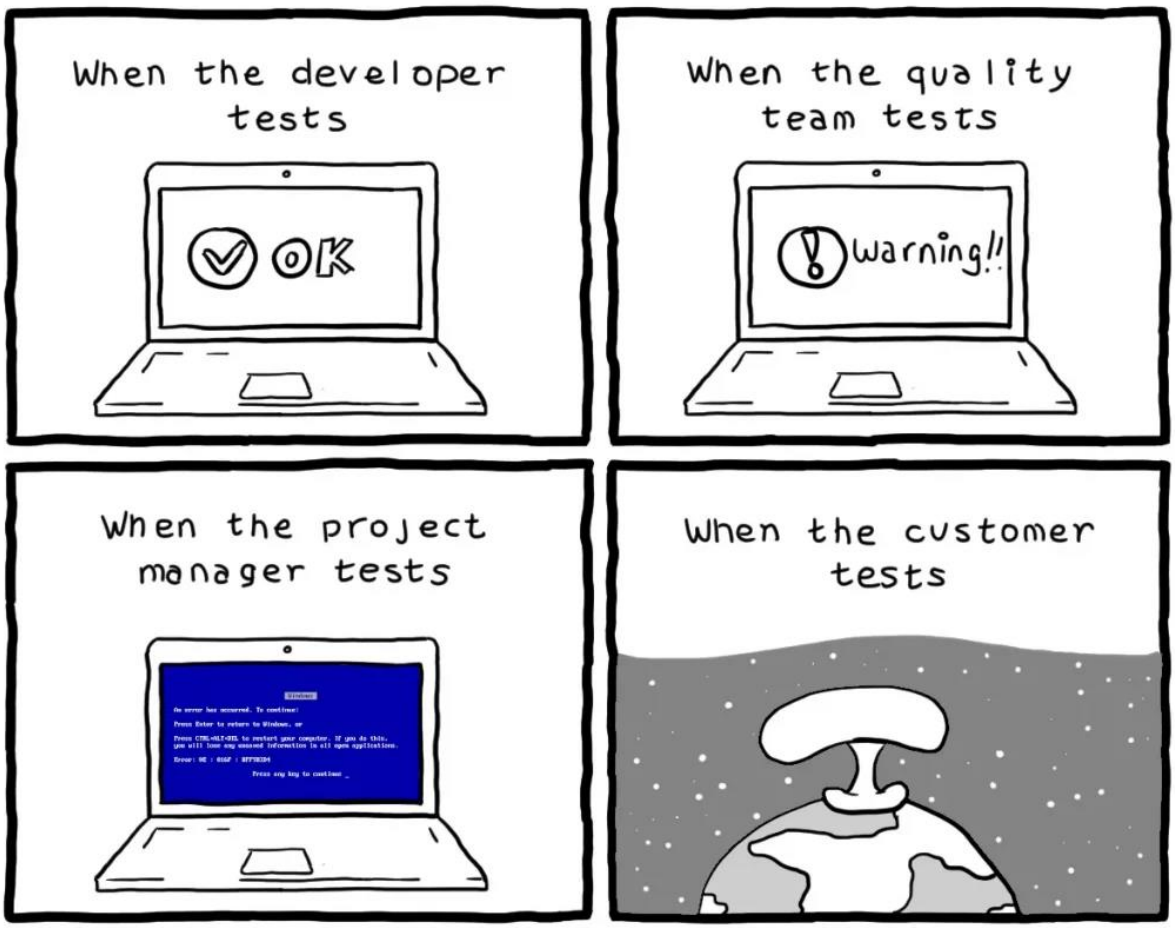
```
import org.junit.Assert;

import org.junit.Test;

public class CalculatorTest {
    @Test
    public void testSuma_2_y_3() {
        //Arrange
        int sumando1 = 2;
        int sumando2 = 3;
        int sumaTotal = 5;
        Calculator calculator = new Calculator();

        //Act
        int resultadoReal = calculator.suma(sumando1, sumando2);

        //Assert
        Assert.assertEquals(sumaTotal, resultadoReal);
    }



    @Test
    public void testSuma_12_y_menos7() {
        //Arrange
        int sumando1 = 12;
        int sumando2 = -7;
        int sumaTotal = 5;
        Calculator calculator = new Calculator();

        //Act
        int resultadoReal = calculator.suma(sumando1, sumando2);

        //Assert
        Assert.assertEquals(sumaTotal, resultadoReal);
    }



    @Test
    public void testResta_12_y_5() {
        //Arrange
        int minuendo = 12;
```

```
        int sustraendo = 5;
        int diferencia = 7;

        //Act
        Calculator calculator = new Calculator();
        int resultadoReal = calculator.resta(minuendo, sustraendo);

        //Assert
        Assert.assertEquals(diferencia, resultadoReal);
    }

    @Test
    public void testResta_7_y_menos7() {
        //Arrange
        int minuendo = 7;
        int sustraendo = -7;
        int diferencia = 14;

        //Act
        Calculator calculator = new Calculator();
        int resultadoReal = calculator.resta(minuendo, sustraendo);

        //Assert
        Assert.assertEquals(diferencia, resultadoReal);
    }



    @Test
    public void testMultiplicacion_7_y_9() {
        //Arrange
        int factor1 = 7;
        int factor2 = 9;
        int producto = 63;

        //Act
        Calculator calculator = new Calculator();
        int resultadoReal = calculator.multiplica(factor1, factor2);

        //Assert
        Assert.assertEquals(producto, resultadoReal);
    }

    @Test
    public void testMultiplicacion_5_y_2() {
        //Arrange
```

```
        int factor1 = 5;
        int factor2 = 2;
        int producto = 10;

        //Act
        Calculator calculator = new Calculator();
        int resultadoReal = calculator.multiplica(factor1, factor2);

        //Assert
        Assert.assertEquals(producto, resultadoReal);
    }


    @Test
    public void testDivision_27_y_3() {
        //Arrange
        int dividendo = 27;
        int divisor = 3;
        int cociente = 9;

        //Act
        Calculator calculator = new Calculator();
        int resultadoReal = calculator.divide(dividendo, divisor);

        //Assert
        Assert.assertEquals(cociente, resultadoReal);
    }

    @Test
    public void testDivision_84_y_7() {
        //Arrange
        int dividendo = 84;
        int divisor = 7;
        int cociente = 12;

        //Act
        Calculator calculator = new Calculator();
        int resultadoReal = calculator.divide(dividendo, divisor);

        //Assert
        Assert.assertEquals(cociente, resultadoReal);
    }
}
```

Ejercicio 2: Testing y Debugging

En éste ejercicio realizaremos pruebas, descubriremos errores y los solucionaremos realizando depuración.



Archivo ComparacionesEnteros

Función sonIguales

Esta función compara dos números y nos dice si son iguales o no. Código:

```
public static boolean sonIguales(int a, int b) {
        boolean resultado = (a != b);
        return resultado;
    }
```

Los test correspondientes para probar esta función son los siguientes:

```
@Test
    public void testSonIgualesV1() {
        int a = 2;
        int b = 4;
        boolean resultadoEsperado = false;

        boolean resultadoReal = ComparacionesEnteros.sonIguales(a, b);

        Assert.assertEquals(resultadoEsperado, resultadoReal);
    }

    @Test
    public void testSonIgualesV2() {
        int a = 4;
        int b = 4;
        boolean resultadoEsperado = true;

        boolean resultadoReal = ComparacionesEnteros.sonIguales(a, b);

        Assert.assertEquals(resultadoEsperado, resultadoReal);
    }
```

Al ejecutarlos, la función no pasa los test. La corregiré para que esté correcta:

```
public static boolean sonIguales(int a, int b) {
        boolean resultado = (a == b);
        return resultado;
    }
```

Si ejecutamos ahora los test, comprobaremos que la función los pasa. El error estaba en la comparación, que en lugar de comprobar si eran iguales, comprobaba si eran distintos.

Función esMayor

Esta función compara dos números y nos dice si el primero es mayor que el segundo. Código:

```
public static boolean esMayor(int a, int b) {
        boolean resultado = (a > b);
        return resultado;
    }
```

Los test correspondientes para probar esta función son los siguientes:

```
@Test
    public void testésMayorV1() {
        int a = 5;
        int b = 2;
        boolean resultadoEsperado = true;

        boolean resultadoReal = ComparacionesEnteros.esMayor(a, b);

        Assert.assertEquals(resultadoEsperado, resultadoReal);
    }

    @Test
    public void testésMayorV2() {
        int a = 7;
        int b = 15;
        boolean resultadoEsperado = false;

        boolean resultadoReal = ComparacionesEnteros.esMayor(a, b);

        Assert.assertEquals(resultadoEsperado, resultadoReal);
    }
```

Al ejecutarlos, la función pasa los test correctamente.

Función esMenor

Esta función compara dos números y nos dice si el primero es menor que el segundo. Código:

```
public static boolean esMenor(int a, int b) {
        boolean resultado = (a > b);
        return resultado;
    }
```

Los test correspondientes para probar esta función son los siguientes:

```
@Test
    public void testésMenorV1() {
        int a = -5;
        int b = 17;
        boolean resultadoEsperado = true;

        boolean resultadoReal = ComparacionesEnteros.esMenor(a, b);

        Assert.assertEquals(resultadoEsperado, resultadoReal);
    }

    @Test
    public void testésMenorV2() {
        int a = 18;
        int b = 10;
        boolean resultadoEsperado = false;

        boolean resultadoReal = ComparacionesEnteros.esMenor(a, b);

        Assert.assertEquals(resultadoEsperado, resultadoReal);
    }
```

Al ejecutarlos, la función no pasa los test. La corregiré para que esté correcta:

```
public static boolean esMenor(int a, int b) {
        boolean resultado = (a < b);
        return resultado;
    }
```

Si ejecutamos ahora los test, comprobaremos que la función los pasa. El error estaba en la comparación, que en lugar de comprobar si el primero era menor, comprobaba si era mayor.

Función esDivisible

Esta función compara dos números y nos dice si el segundo es divisible del primero.
Código:

```
public static boolean esDivisible(int a, int b) {
        boolean divisorEsCero = (b == 0);
        boolean resultado;
        if (divisorEsCero) {
            resultado = true;
        } else {
            int residuo = a % b;
            resultado = (residuo != 0);
        }
        return resultado;
    }
```

Los test correspondientes para probar esta función son los siguientes:

```
@Test
    public void testésDivisibleV1() {
        int a = 4;
        int b = 2;
        boolean resultadoEsperado = true;

        boolean resultadoReal = ComparacionesEnteros.esDivisible(a, b);

        Assert.assertEquals(resultadoEsperado, resultadoReal);
    }

    @Test
    public void testésDivisibleV2() {
        int a = 52;
        int b = 3;
        boolean resultadoEsperado = false;

        boolean resultadoReal = ComparacionesEnteros.esDivisible(a, b);

        Assert.assertEquals(resultadoEsperado, resultadoReal);
    }

    @Test
    public void testésDivisibleV3() {
        int a = 52;
        int b = 0;
        boolean resultadoEsperado = false;

        boolean resultadoReal = ComparacionesEnteros.esDivisible(a, b);
```

```
        Assert.assertEquals(resultadoEsperado, resultadoReal);
    }
```

Al ejecutarlos, la función no pasa los test. La corregiré para que esté correcta:

```
public static boolean esDivisible(int a, int b) {
        boolean divisorEsCero = (b == 0);
        boolean resultado;
        if (divisorEsCero) {
            resultado = false;
        } else {
            int residuo = a % b;
            resultado = (residuo == 0);
        }
        return resultado;
    }
```

Si ejecutamos ahora los test, comprobaremos que la función los pasa. El código tenía 2 errores:

1. Un número no puede ser divisible entre 0, sin embargo, la función nos devolvía “true” cuando el divisor era 0. Había que devolver “false”.
2. Cuando comparaba el resultado con el resto, comprobaba si el resto era distinto a 0 en lugar de igual a 0. Había que comprobar si era 0.

Función sonAmbosPares

Esta función comprueba dos números y nos dice si ambos son pares. Código:

```
public static boolean sonAmbosPares(int a, int b) {
        boolean primerNumeroPar = (a % 2 == 0);
        boolean segundoNumeroPar = (b % 2 == 0);
        boolean resultado = primerNumeroPar || segundoNumeroPar;
        return resultado;
    }
```

Los test correspondientes para probar esta función son los siguientes:

```
@Test
    public void testSonAmbosParesV1() {
        int a = 18;
        int b = 10;
        boolean resultadoEsperado = true;

        boolean resultadoReal = ComparacionesEnteros.sonAmbosPares(a, b);

        Assert.assertEquals(resultadoEsperado, resultadoReal);
    }

    @Test
    public void testSonAmbosParesV2() {
        int a = 6;
        int b = 1;
        boolean resultadoEsperado = false;

        boolean resultadoReal = ComparacionesEnteros.sonAmbosPares(a, b);

        Assert.assertEquals(resultadoEsperado, resultadoReal);
    }
```

Al ejecutarlos, vemos que no pasa el segundo. La corregiré para que esté correcta:

```
 public static boolean sonAmbosPares(int a, int b) {
        boolean primerNumeroPar = (a % 2 == 0);
        boolean segundoNumeroPar = (b % 2 == 0);
        boolean resultado = primerNumeroPar && segundoNumeroPar;
        return resultado;
    }
```

Si ejecutamos ahora los test, si los pasa. Solo había que cambiar el operador lógico “OR” por “AND” al darle un valor a la variable “resultado”.

Función alMenosUnoPositivo

Esta función comprueba dos números y nos dice si hay algún positivo. Código:

```
public static boolean alMenosUnoPositivo(int a, int b) {
        boolean primerNumeroPositivo = (a > 0);
        boolean segundoNumeroPositivo = (b < 0);
        boolean resultado = primerNumeroPositivo && segundoNumeroPositivo;
        return resultado;
    }
```

Los test correspondientes para probar esta función son los siguientes:

```
@Test
    public void testAlMenosUnoPositivoV1() {
        int a = -5;
        int b = 2;
        boolean resultadoEsperado = true;

        boolean resultadoReal = ComparacionesEnteros.alMenosUnoPositivo(a, b);

        Assert.assertEquals(resultadoEsperado, resultadoReal);
    }

    @Test
    public void testAlMenosUnoPositivoV2() {
        int a = -5;
        int b = -2;
        boolean resultadoEsperado = false;

        boolean resultadoReal = ComparacionesEnteros.alMenosUnoPositivo(a, b);

        Assert.assertEquals(resultadoEsperado, resultadoReal);
    }
```

Al ejecutarlos, vemos que no pasa el segundo. La corregiré para que esté correcta:

```
public static boolean alMenosUnoPositivo(int a, int b) {
        boolean primerNumeroPositivo = (a > 0);
        boolean segundoNumeroPositivo = (b > 0);
        boolean resultado = primerNumeroPositivo || segundoNumeroPositivo;
        return resultado;
    }
```

Tenia dos errores. El primero, al dar valor a la segunda variable, comprobaba si era menor a cero. El segundo, había que cambiar el operador lógico “AND” por “OR”.

Función sumaEsPar

Esta función comprueba dos números y nos dice si ambos son pares. Código:

```
public static boolean sumaEsPar(int a, int b) {
        int suma = ++a + b;
        boolean sumaPar = (++suma % 2 == 0);
        return sumaPar;
    }
```

Los test correspondientes para probar esta función son los siguientes:

```
@Test
    public void testSumaEsParV1() {
        int a = 11;
        int b = 23;
        boolean resultadoEsperado = true;

        boolean resultadoReal = ComparacionesEnteros.sumaEsPar(a, b);

        Assert.assertEquals(resultadoEsperado, resultadoReal);
    }

    @Test
    public void testSumaEsParV2() {
        int a = 12;
        int b = 23;
        boolean resultadoEsperado = false;

        boolean resultadoReal = ComparacionesEnteros.sumaEsPar(a, b);

        Assert.assertEquals(resultadoEsperado, resultadoReal);
    }
```

Aunque al ejecutarlos, la función pase ambos test, la manera correcta de codificar esta función es la siguiente:

```
public static boolean sumaEsPar(int a, int b) {
        int suma = a + b;
        boolean sumaPar = (suma % 2 == 0);
        return sumaPar;
    }
```

Archivo OperacionesMixtas

Función concatenarNumeroTexto

Esta función pide un número y un texto, y los concatena con un espacio. Código:

```
public static String concatenarNumeroTexto(int numero, String texto) {
        String numeroComoTexto = Integer.toString(numero);
        String resultado = numeroComoTexto + " " + numero;
        return resultado;
    }
```

Los test correspondientes para probar esta función son los siguientes:

```
    @Test
    public void testConcatenarNumeroTextoV1() {
        int numero = 5;
        String texto = "manzanas";
        String resultadoEsperado = "5 manzanas";

        String resultadoReal = OperacionesMixtas.concatenarNumeroTexto(numero,
texto);

        Assert.assertEquals(resultadoEsperado, resultadoReal);
    }

    @Test
    public void testConcatenarNumeroTextoV2() {
        int numero = 8;
        String texto = "manzanas rojas";
        String resultadoEsperado = "8 manzanas rojas";

        String resultadoReal = OperacionesMixtas.concatenarNumeroTexto(numero,
texto);

        Assert.assertEquals(resultadoEsperado, resultadoReal);
    }
```

Al ejecutarlos, la función no pasa los test. La corregiré para que esté correcta:

```
public static String concatenarNumeroTexto(int numero, String texto) {
        String numeroComoTexto = Integer.toString(numero);
        String resultado = numeroComoTexto + " " + texto;
        return resultado;
    }
```

Solo había que cambiar la variable “resultado”, ya que concatenaba el número en lugar del texto.

Función booleanComoTexto

Esta función pide un booleano, y devuelve “verdadero” o “falso”. Código:

```
public static String booleanComoTexto(boolean valor) {
        String resultado;
        if (valor) {
            resultado = "verdadero";
        } else {
            resultado = "verdadero";
        }
        return resultado;
    }
```

Los test correspondientes para probar esta función son los siguientes:

```
@Test
    public void testBooleanComoTextoV1() {
        boolean valor = true;
        String resultadoEsperado = "verdadero";

        String resultadoReal = OperacionesMixtas.booleanComoTexto(valor);

        Assert.assertEquals(resultadoEsperado, resultadoReal);
    }
    @Test
    public void testBooleanComoTextoV2() {
        boolean valor = false;
        String resultadoEsperado = "falso";

        String resultadoReal = OperacionesMixtas.booleanComoTexto(valor);

        Assert.assertEquals(resultadoEsperado, resultadoReal);
    }
```

Al ejecutarlos, la función no pasa el segundo test. La corregiré para que esté correcta:

```
public static String booleanComoTexto(boolean valor) {
        String resultado;
        if (valor) {
            resultado = "verdadero";
        } else {
            resultado = "falso";
        }
        return resultado;
    }
```

El error era el resultado dado en el caso de que “valor” fuera falso, que guardaba “verdadero” en lugar de “falso”.

Función concatenarTextos

Esta función pide dos Strings, y los concatena con un espacio en medio. Código:

```
public static String concatenarTextos(String texto1, String texto2) {
        String resultado = texto1 + "" + texto2;
        return resultado;
    }
```

Los test correspondientes para probar esta función son los siguientes:

```
@Test
    public void testConcatenarTextosV1() {
        String texto1 = "Hola,";
        String texto2 = "Mundo";
        String resultadoEsperado = "Hola, Mundo";

        String resultadoReal = OperacionesMixtas.concatenarTextos(texto1,
texto2);

        Assert.assertEquals(resultadoEsperado, resultadoReal);
    }

    @Test
    public void testConcatenarTextosV2() {
        String texto1 = "He sacado en Entornos de Desarrollo un";
        String texto2 = "10";
        String resultadoEsperado = "He sacado en Entornos de Desarrollo un
10";

        String resultadoReal = OperacionesMixtas.concatenarTextos(texto1,
texto2);

        Assert.assertEquals(resultadoEsperado, resultadoReal);
    }
```

Al ejecutarlos, la función no pasa los test. La corregiré para que esté correcta:

```
    public static String concatenarTextos(String texto1, String texto2) {
        String resultado = texto1 + " " + texto2;
        return resultado;
    }
```

Había que cambiar las comillas al concatenar los textos y colocar un espacio.

Función descripcionNumero

Esta función pide un número y nos dice si es positivo, negativo o 0. Código:

```
public static String descripcionNumero(int numero) {
        String resultado;
        if (numero < 0) {
            resultado = "El número es positivo";
        } else if (numero <= 0) {
            resultado = "El número es negativo";
        } else {
            resultado = "El número es cero";
        }
        return resultado;
    }
```

Los test correspondientes para probar esta función son los siguientes:

```
@Test
    public void testDescripcionNumeroV1() {
        int numero = 5;
        String resultadoEsperado = "El número es positivo";

        String resultadoReal = OperacionesMixtas.descripcionNumero(numero);

        Assert.assertEquals(resultadoEsperado, resultadoReal);
    }

    @Test
    public void testDescripcionNumeroV2() {
        int numero = -7;
        String resultadoEsperado = "El número es negativo";

        String resultadoReal = OperacionesMixtas.descripcionNumero(numero);

        Assert.assertEquals(resultadoEsperado, resultadoReal);
    }

    @Test
    public void testDescripcionNumeroV3() {
        int numero = 0;
        String resultadoEsperado = "El número es cero";

        String resultadoReal = OperacionesMixtas.descripcionNumero(numero);

        Assert.assertEquals(resultadoEsperado, resultadoReal);
    }
```

Al ejecutarlos, la función no pasa los test. La corregiré para que esté correcta:

```
    public static String descripcionNumero(int numero) {
```

```
        String resultado;
        if (numero < 0) {
            resultado = "El número es negativo";
        } else if (numero > 0) {
            resultado = "El número es positivo";
        } else {
            resultado = "El número es cero";
        }
        return resultado;
    }
```

Había que cambiar las condiciones y el valor de “resultado” dependiendo del número.

Función cambiarTextoAMayusculas

Esta función pide un texto y, dependiendo de un booleano, lo convierte a mayúscula o a minúscula. Código:

```
    public static String cambiarTextoAMayusculas(String texto, boolean
aMayusculas) {
        String resultado;
        if (aMayusculas) {
            resultado = texto.toUpperCase();
        } else {
            resultado = texto.toLowerCase();
        }
        return resultado;
    }
```

Los test correspondientes para probar esta función son los siguientes:

```
@Test
    public void testCambiarTextoAMayusculasV1() {
        boolean aMayusculas = true;
        String texto = "Hola";
        String resultadoEsperado = "HOLA";

        String resultadoReal =
OperacionesMixtas.cambiarTextoAMayusculas(texto, aMayusculas);

        Assert.assertEquals(resultadoEsperado, resultadoReal);
    }

    @Test
    public void testCambiarTextoAMayusculasV2() {
        boolean aMayusculas = false;
        String texto = "Hola";
        String resultadoEsperado = "hola";

        String resultadoReal =
OperacionesMixtas.cambiarTextoAMayusculas(texto, aMayusculas);

        Assert.assertEquals(resultadoEsperado, resultadoReal);
    }
```

Al ejecutarlos, la función pasa los test.

Función descripcionConFloat

Esta función pide dos valores "float", los convierte a "String" añadiéndole un símbolo adicional, y luego los concatena con otro texto. Código:

```
public static String descripcionConFloat(float valor1, float valor2) {
        String valor1ComoTexto = Float.toString(valor1) + "€";
        String valor2ComoTexto = Float.toString(valor2) + "€";
        String resultado = "Los valores son: " + valor1ComoTexto + " y " +
valor2ComoTexto;
        return resultado;
    }
```

Los test correspondientes para probar esta función son los siguientes:

```
    @Test
    public void testDescripcionConFloatV1() {
        float valor1 = 3.14f;
        float valor2 = 2.71f;
        String resultadoEsperado = "Los valores son: 3.14€ y 2.71€";

        String resultadoReal = OperacionesMixtas.descripcionConFloat(valor1,
valor2);

        Assert.assertEquals(resultadoEsperado, resultadoReal);
    }

    @Test
    public void testDescripcionConFloatV2() {
        float valor1 = -3.32f;
        float valor2 = -76.34f;
        String resultadoEsperado = "Los valores son: -3.32€ y -76.34€";

        String resultadoReal = OperacionesMixtas.descripcionConFloat(valor1,
valor2);

        Assert.assertEquals(resultadoEsperado, resultadoReal);
    }
```

Al ejecutarlos, la función pasa los test.

Función formatearNumero

Esta función un "float", lo convierte a "String" con el número de decimales que se le pase como segundo parámetro. Código:

```
public static String formatearNumero(float numero, int decimales) {
        String numeroFormateado = String.format("%."+decimales+"f", numero);
        String resultado = numeroFormateado;
        return resultado;
    }
```

Los test correspondientes para probar esta función son los siguientes:

```
    @Test
    public void testFormatearNumeroV1() {
        float numero = 17.89945f;
        int decimales = 2;
        String resultadoEsperado = "17,90";

        String resultadoReal = OperacionesMixtas.formatearNumero(numero,
decimales);

        Assert.assertEquals(resultadoEsperado, resultadoReal);
    }

    @Test
    public void testFormatearNumeroV2() {
        float numero = 15.556f;
        int decimales = 2;
        String resultadoEsperado = "15,56";

        String resultadoReal = OperacionesMixtas.formatearNumero(numero,
decimales);

        Assert.assertEquals(resultadoEsperado, resultadoReal);
    }
```

Al ejecutarlos, la función pasa los test.