

Diseño y realización de pruebas

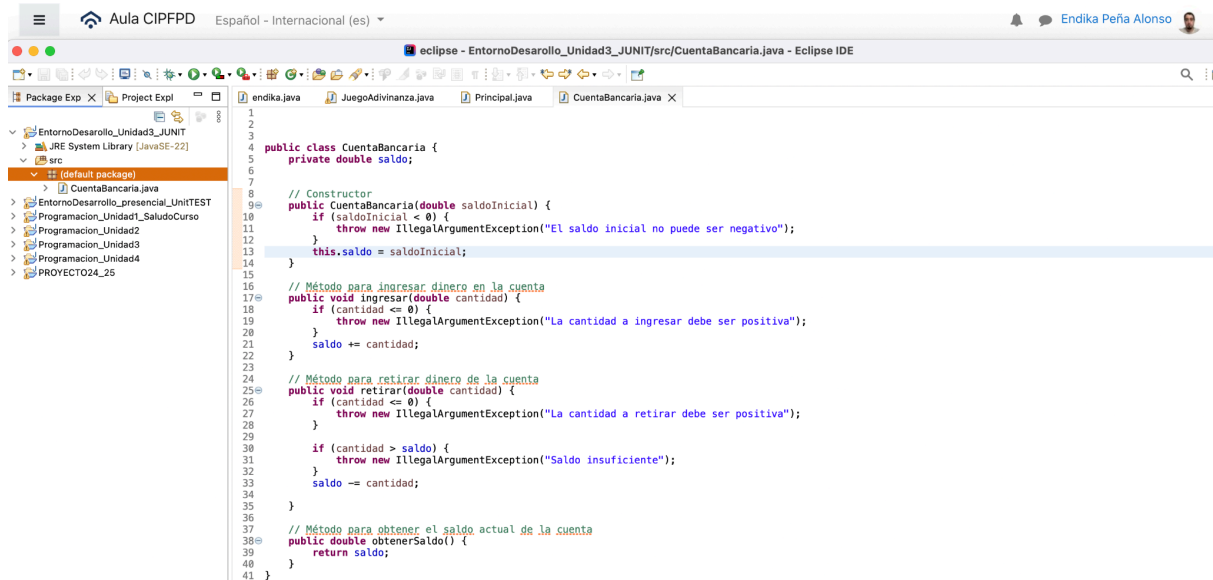
TAREA 3

Índice

| | |
|---------------------------------|---|
| 1. Pruebas unitarias Junit..... | 3 |
| Pasos a seguir:..... | 3 |
| 2. Depuración en Eclipse..... | 6 |

1. Pruebas unitarias Junit

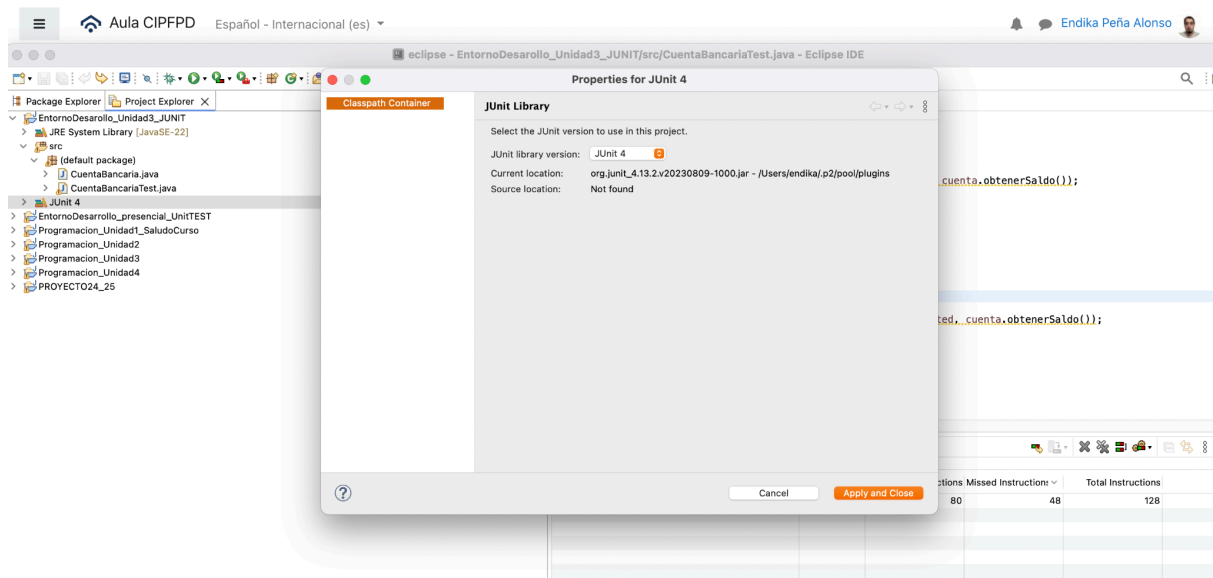
Creamos primero el código Java del que vamos a hacer los test.



```
1
2
3
4 public class CuentaBancaria {
5     private double saldo;
6
7
8     // Constructor
9     public CuentaBancaria(double saldoInicial) {
10         if (saldoInicial < 0) {
11             throw new IllegalArgumentException("El saldo inicial no puede ser negativo");
12         }
13         this.saldo = saldoInicial;
14     }
15
16     // Método para ingresar dinero en la cuenta
17     public void ingresar(double cantidad) {
18         if (cantidad <= 0) {
19             throw new IllegalArgumentException("La cantidad a ingresar debe ser positiva");
20         }
21         saldo += cantidad;
22     }
23
24     // Método para retirar dinero de la cuenta
25     public void retirar(double cantidad) {
26         if (cantidad <= 0) {
27             throw new IllegalArgumentException("La cantidad a retirar debe ser positiva");
28         }
29         if (cantidad > saldo) {
30             throw new IllegalArgumentException("Saldo insuficiente");
31         }
32         saldo -= cantidad;
33     }
34
35     // Método para obtener el saldo actual de la cuenta
36     public double obtenerSaldo() {
37         return saldo;
38     }
39
40
41 }
```

Pasos a seguir:

- Creación de clase de test CuentaBancariaTest.java dentro del mismo paquete.



- Inicializa la cuenta bancaria que vas a utilizar para las pruebas con un saldo de 6.000€ utilizando el constructor CuentaBancaria.

- **testObtenerSaldo()**: Verifica que el saldo inicial sea 6.000€.

```
@Test
public void testObtenerSaldo() {
    CuentaBancaria cuenta = new CuentaBancaria(6000);
    double saldo = cuenta.obtenerSaldo();

    double expected = 6000;

    assertEquals("El resultado esperado es 6000", expected, saldo, 0.00001);
}
```

- **testIngresarDinero()**: Verifica que al ingresar dinero, el saldo aumente correctamente.

```
@Test
public void testIngresarDinero() {
    CuentaBancaria cuenta = new CuentaBancaria(6000);
    cuenta.ingresar(20);

    double saldo = cuenta.obtenerSaldo();

    double expected = 6020;

    assertEquals("El resultado esperado es 6020", expected, saldo, 0.00001);
}
```

- **testRetirarDinero()**: Verifica que al retirar dinero, el saldo disminuya correctamente.

```
@Test
public void testRetirarDinero() {
    CuentaBancaria cuenta = new CuentaBancaria(6000);
    cuenta.retirar(20);

    double saldo = cuenta.obtenerSaldo();

    double expected = 5980;

    assertEquals("El resultado esperado es 5980", expected, saldo, 0.00001);
}
```

- **testRetiroExcesivo()**: Verifica que lanzar una excepción si se intenta retirar más dinero del que hay en la cuenta.

```
@Test
public void testRetiroExcesivo() {
    CuentaBancaria cuenta = new CuentaBancaria(20);

    try {
        cuenta.retirar(30); // Esto debería lanzar una excepción
        fail("Se esperaba una IllegalArgumentException pero no se lanzó."); // Marca la prueba como fallida si no se lanza la excepción
    } catch (IllegalArgumentException e) {
        System.out.println(e.getMessage());
        assertEquals("Saldo insuficiente", e.getMessage()); // Verifica el mensaje de la excepción
    }
}
```

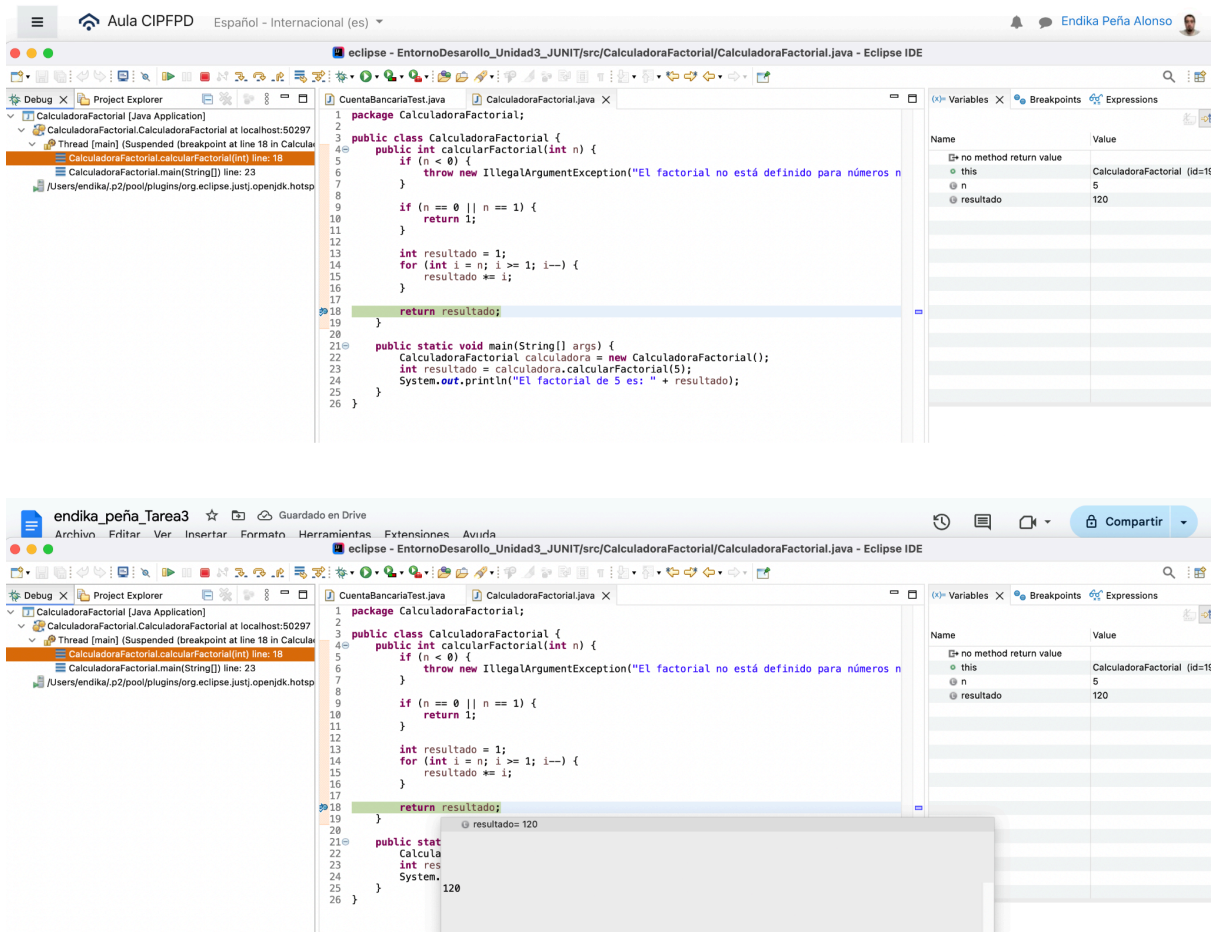
- **testIngresoNegativo()**: Verifica que lanzar una excepción si se intenta ingresar una cantidad negativa.

```
@Test
public void testIngresoNegativo() {
    CuentaBancaria cuenta = new CuentaBancaria(20);

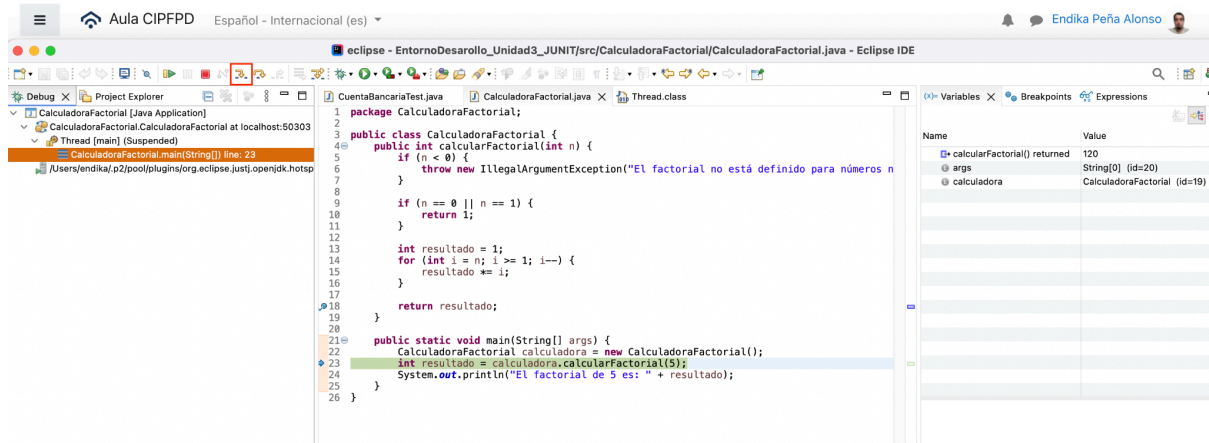
    try {
        cuenta.ingresar(-130); // Esto debería lanzar una excepción
        fail("Se esperaba una IllegalArgumentException pero no se lanzó."); // Marca la prueba como fallida si no se lanza la excepción
    } catch (IllegalArgumentException e) {
        System.out.println(e.getMessage());
        assertEquals("La cantidad a ingresar debe ser positiva", e.getMessage()); // Verifica el mensaje de la excepción
    }
}
```

2. Depuración en Eclipse

Aquí hay un código Java con un pequeño error, utiliza la depuración para testearlo.



Para realizar la ejecución paso a paso después de darle a debug as debemos pulsar sobre el botón que he marcado en la imagen para poder ir paso a paso haciendo seguimiento de lo que hace el código.



En este caso he buscado el error pero no lo estoy viendo debido a que el resultado factorial de 5 es 120 y el resultado del programa es 120, no veo donde está el error.