

Refactorització i Proves Unitàries d'un sistema bancari bàsic

Part 1: Depuració del codi

1. Descripció inicial del codi:

- **Què fa el codi? (Explicar breument).**

El codi gestiona una factura, permetent calcular el subtotal, els impostos i el total, així com mostrar els resultats.

A més, des del fitxer Main, es crea una instància de Factura, es defineix el subtotal, es calculen els impostos i el total, i es mostra la factura resultant.

- **Quins són els mètodes més importants i què fan?**

1. **establecerSubtotal(double subtotal):** Asigna el subtotal de la factura.
2. **calcularImpuestos(double porcentajeImpuesto):** Calcula los impuestos aplicables basados en el porcentaje proporcionado.
3. **calcularTotal():** Suma el subtotal y los impuestos para calcular el total.
4. **mostrarFactura():** Muestra el subtotal, los impuestos y el total.

- **Quin és el valor inicial del saldo (balance) abans de realitzar qualsevol operació?**

Els valors inicials són:

subtotal: 0.0

impuestos: 0.0

total: 0.0

2. Posar punts de control (Breakpoints):

- Per depurar el codi, utilitza els punts de control (breakpoints). Això permet aturar l'execució del codi en determinats punts i examinar l'estat de les variables. Per afegir un punt de control, fes clic a la barra de l'esquerra de la línia on vols aturar el codi.
- On has col·locat els punts de control (breakpoints) i per què?

Dins de RefactorExample

Main:

- **Línia on es crea l'objecte Factura** → Per verificar que s'inicialitza correctament.

```
8 Factura factura = new Factura();
```

- **Línies on es criden calcularImpuestos() i calcularTotal()** → Per comprovar que els càlculs es fan correctament.

```
10 factura.calcularImpuestos(21); // 21% de IVA
```

- **Línia on es crida mostrarFactura()** → Per veure els valors finals.

```
12 factura.mostrarFactura();
```

Factura:

- **Dins de calcularImpuestos()** → Per veure el valor de porcentajelimpuesto i this.impuestos.

```
18 System.out.println("Impuestos: " + this.impuestos);
```

- **Dins de calcularTotal()** → Per confirmar que total es calcula correctament.

```
19 System.out.println("Total: " + this.total);
```

Dins de SimpleBankingSystem

Main:

- **Línia on es crea l'objecte Account** → Per verificar que els valors s'estableixen correctament.

```
10 myAccount = new Account("Flor Martinez", "1000-1234-56-123456789",
```

- **Línia on es crida withdrawAmount(2300)** → Per veure si es gestiona bé el saldo.

```
13 myAccount.withdrawAmount(2300);
```

- **Línia on es crida depositAmount(1695)** → Per assegurar que el mètode funciona correctament.

```
21 myAccount.depositAmount(1695);
```

Account:

- **Dins de withdrawAmount(), just abans de fer la resta** → Per veure el valor de amount i balance.

```
63 if (getBalance() < amount)
64     throw new Exception("No hi ha suficient saldo");
65     balance -= amount;
66 }
```

- **Dins de depositAmount(), abans de modificar balance** → Per confirmar que amount és correcte.

```
51 if (amount < 0)
52     throw new Exception("No es pot ingressar una quantitat negativa");
53     balance += amount;
54 }
```

3. Examina les variables i el flux d'execució:

- A mesura que el codi s'atura a cada punt de control, observa el valor de les variables name, account i balance. Inclou una captura de

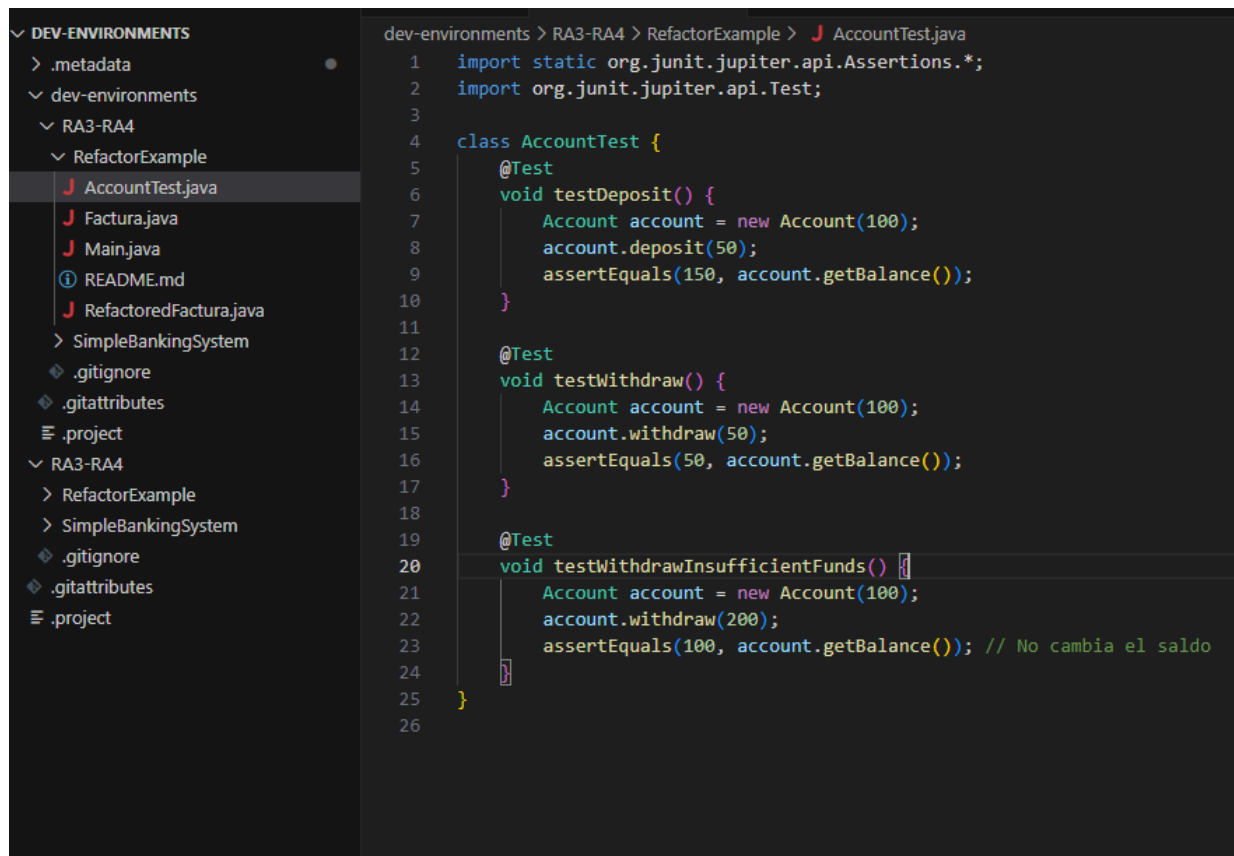
pantalles dels valors de les variables a mesura que avancen les operacions.

4. Explora les excepcions:

- Feu els canvis necessaris al Main per fer saltar les excepcions. Inclou la captura de pantalla d'un missatge d'error generat per una excepció i com es visualitza al terminal o a la consola de Eclipse.

Part 3: Proves Unitàries

1. Crear una classe de proves que es digui “AccountTest” i contingui dos mètodes: un per provar els dipòsits i un altre per provar les retirades.
2. Les proves han de verificar casos d'èxit, així com casos amb errors (quantitat negativa, saldo insuficient). Utilitzeu el mètode assertEquals.



The screenshot shows an IDE with a project structure on the left and the code of `AccountTest.java` on the right. The project structure includes:

- DEV-ENVIRONMENTS
 - .metadata
 - dev-environments
 - RA3-RA4
 - RefactorExample
 - AccountTest.java**
 - Factura.java
 - Main.java
 - README.md
 - RefactoredFactura.java
 - SimpleBankingSystem
 - .gitignore
 - .gitattributes
 - .project
 - RA3-RA4
 - RefactorExample
 - SimpleBankingSystem
 - .gitignore
 - .gitattributes
 - .project

The code in `AccountTest.java` is as follows:

```
dev-environments > RA3-RA4 > RefactorExample > J AccountTest.java
1  import static org.junit.jupiter.api.Assertions.*;
2  import org.junit.jupiter.api.Test;
3
4  class AccountTest {
5      @Test
6      void testDeposit() {
7          Account account = new Account(100);
8          account.deposit(50);
9          assertEquals(150, account.getBalance());
10     }
11
12     @Test
13     void testWithdraw() {
14         Account account = new Account(100);
15         account.withdraw(50);
16         assertEquals(50, account.getBalance());
17     }
18
19     @Test
20     void testWithdrawInsufficientFunds() {
21         Account account = new Account(100);
22         account.withdraw(200);
23         assertEquals(100, account.getBalance()); // No cambia el saldo
24     }
25 }
26
```

<https://github.com/josueiranzo/dev-environments.git>