

Refactorització i Proves Unitàries d'un sistema bancari bàsic**Resultats d'aprenentatge:**

RA3. Verifica el funcionament de programes dissenyant i realitzant proves.

RA4. Optimitza codi emprant les eines disponibles en l'entorn de desenvolupament.

Objectiu de la pràctica:

- Aprendre a depurar i analitzar el codi per comprendre com funcionen les variables i la seva interacció en el programa.
 - Refactoritzar el codi separant les responsabilitats mitjançant la tècnica de Extract Method.
 - Escriure proves unitàries per assegurar el correcte funcionament dels mètodes.
 - Utilitzar GitHub per gestionar el control de versions mitjançant branques.
-

Instruccions d'ús:

1. Creeu un nou repositori pel mòdul.
 2. Configurar Eclipse per treballar amb Git:
Abans de començar, assegura't de tenir instal·lada la funcionalitat de Git en Eclipse. Si no tens el plugin de Git, pots instal·lar-ho seguint aquests passos:
 - Ves a Help > Eclipse Marketplace.
 - Busca EGit (que és el plugin de Git per Eclipse) i instal·la'l si no el tens.
 3. Clonar el repositori original des d'Eclipse:
<https://github.com/fmartinez-dev-learn/dev-environments>
→ RA3-RA4/SimpleBankingSystem
 4. Al mateix repositori podeu trobar un exemple de refactorització al directori
→ RA3-RA4/Refactor exemple
 5. Show in local terminal per treballar des de l'eclipse amb git.
-

Instruccions d'entrega:

- Repositori de GitHub.
- README actualitzat.
- Documentació PDF.

Part 1: Depuració del Codi

Objectiu: entendre com es gestionen les operacions de dipositar i retirar, com es modifiquen les variables i com es capturen les excepcions en cas d'errors.

Documenteu els següents passos:

1. Descripció inicial del codi:

- Què fa el codi? (Explicar breument).

Para mirar el saldo actual de una persona, que tienes en la cuenta bancaria.

Escribiendo: el nombre, el numero de la cuenta y el saldo que se tiene al comienzo de todo. Después hace un retiro y la cuenta queda en 1895€.

- Quins són els mètodes més importants i què fan?

Los metodos más importantes son depositAmount que sirve para depositar el dinero de la persona y whithDrawAmount que es el metodo para retirar dinero .

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

El saldo inicial de la persona es de 2500 €.

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?
- Inclou una captura de pantalla de Eclipse amb els breakpoints activats abans de començar la depuració.

CLASE MAIN :

BREAKPOINTS :

1.- Línea 10 :Este es el momento en el que se inicializa la cuenta bancaria con los

datos , nombre, numero de cuenta y saldo inicial .

Con este breakpoint podemos inspeccionar si los valores pasados al constructor se estan agregando correctamente .

2.- Linea 12,13,14,15,16 : en estas lineas comprobamos el retiro de dinero de la cuenta , si el metodo withdrawAmount es invocado correctamente con el monto proporcionado , si esta correcto nos mostrara si hay saldo suficiente o si el monto es valido , si hay un error se lanzara alguna excepcion .

3.-Line 19 , 20, 21,22,23 :Antes de realizar el deposito confirmamos que el flujo de la operacion pasa correctamente .En myAccount.depositAmount(1695); verificamos si el metodo verifica correctamente el depósito para un deposito valido, tambien comprueba que el saldo cambie despues de la operaciò.

En el catch inspeccionamos si las excepciones lanzadas por el metodo son correctamente manejadas , si todo va bien lanzara el mensaje "El sueldo actual es".

4.-

```

6  > public class main {
7  >     public static void main(String[] args) {
8      Account myAccount;
9
10     myAccount = new Account( name: "Flor Martinez", account: "1000-1234-56-123456789", balance: 2500);
11
12     try {
13         myAccount.withdrawAmount(2300);
14     } catch(Exception e){
15         System.err.println(e.getMessage());
16         System.out.println("Error al retirar");
17     }
18
19     try {
20         System.out.println("Ingrés al compte");
21         myAccount.depositAmount(1695);
22     } catch(Exception e){
23         System.err.println(e.getMessage());
24         System.out.println("Error en l'ingrés");
25     }
26
27     System.out.println("El saldo actual es " + myAccount.getBalance());
28 }
29 }
30

```

CLASE ACCOUNT:

```

14 public Account() {
15     super();
16 }
17
18 //constructor amb arguments
19 3 usages
20 public Account(String name, String account, double balance) {
21     super();
22     this.name = name;
23     this.account = account;
24     this.balance = balance;
25 }
26
27 //mètode per tornar el nom del titular del compte
28 no usages
29 public String getName() {
30     return name;
31 }
32
33 //mètode per actualitzar el nom del titular del compte
34 no usages
35 public void setName(String name) {
36     this.name = name;
37 }
38
39 //mètode per tornar el numero de compte
40 no usages
41 public String getAccount() {
42     return account;
43 }

```

Línea 14

Para asegurarte de que el constructor por defecto se está ejecutando correctamente y que la llamada al constructor de la clase base (super()) funciona como se espera.

Línea 19

Para verificar que el constructor con argumentos también realiza correctamente la llamada al constructor de la clase base.

Línea 20

Para confirmar que el parámetro name se asigna correctamente a la variable miembro name.

Línea 21

Para asegurar que el parámetro account se asigna correctamente a la variable miembro account.

Línea 22

Para verificar que el parámetro balance se asigna correctamente a la variable miembro balance.

Línea 27

Para comprobar que el método getName devuelve correctamente el valor de la variable name.

Línea 32

Para asegurarte de que el método setName actualiza correctamente la variable miembro name.

Línea 37

Para confirmar que el método getAccount devuelve correctamente el valor de la

variable account.

```
33 public void setName(String name) {  
34     this.name = name;  
35 }  
36 //mètode per tornar el numero de compte  
37 no usages  
38 public String getAccount() {  
39     return account;  
40 }  
41 //mètode per tornar el saldo disponible del compte  
42 4 usages  
43 public double getBalance() {  
44     return balance;  
45 }  
46 /* mètode per ingressar quantitats al compte  
47  * modifica el saldo  
48  */  
49 3 usages  
50 public void depositAmount(double amount) throws Exception  
51 {  
52     if (amount<0)  
53         throw new Exception("No es pot ingressar una quantitat negativa.");  
54     balance += amount;  
55 }
```

Línea 33

Para asegurarte de que el método setName está actualizando correctamente la variable miembro name.

Línea 37

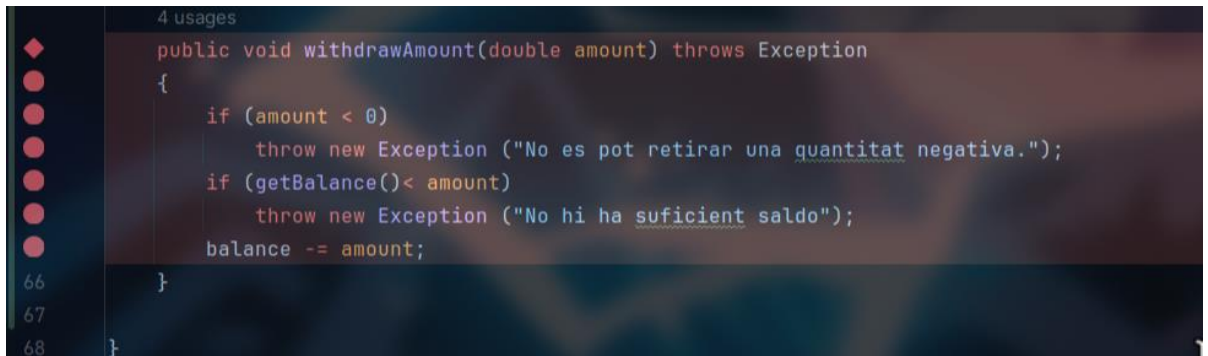
Para verificar que el método getAccount devuelve correctamente el valor de la variable account.

Línea 42

Para confirmar que el método getBalance devuelve correctamente el valor de la variable balance.

Línea 49

Para comprobar que el método depositAmount modifica correctamente el saldo de la cuenta, especialmente cuando se ingresa una cantidad positiva.



```
4 usages
public void withdrawAmount(double amount) throws Exception
{
    if (amount < 0)
        throw new Exception ("No es pot retirar una quantitat negativa.");
    if (getBalance() < amount)
        throw new Exception ("No hi ha suficient saldo");
    balance -= amount;
}
```

Línea 66

Para verificar si el amount (monto) es negativo y si se está cumpliendo la condición.

Línea 67

Para confirmar que la excepción se lanza correctamente cuando el monto es negativo.

Línea 68

Para comprobar si el saldo actual es menor que el monto que se quiere retirar y si se está cumpliendo la condición.

Línea 69

Para asegurarte de que la excepción se lanza correctamente cuando el saldo es insuficiente.

Línea 70

Para verificar que el saldo se actualiza correctamente cuando el retiro es válido y que la cantidad se resta del balance.

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.

1.-

```

public Account(String name, String account, double balance) {
    super();
    this.name = name;
    this.account = account;
    this.balance = balance;
}

```

2.-

```

public double getBalance() {
    return balance;
}

```

4. Explora les excepcions:

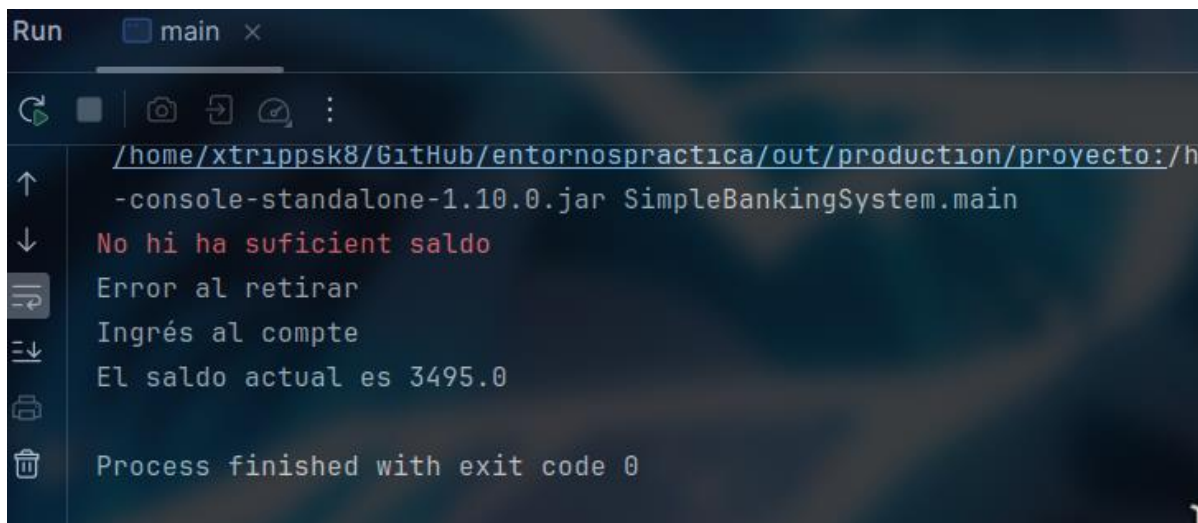
- Feu els canvis necessaris al `Main` per fer saltar les excepcions. Inclou la captura de pantalla d'un missatge d'error generat per una excepció i com es visualitza al terminal o a la consola de Eclipse.

```

package SimpleBankingSystem;
/**
 *
 * @author Flor Martinez
 */
public class main {
    public static void main(String[] args) {
        Account myAccount;
        myAccount = new Account("Flor Martinez", "1000-1234-56-123456789", 1800);
        try {
            myAccount.withdrawAmount(2300);
        } catch (Exception e) {
            System.err.println(e.getMessage());
            System.out.println("Error al retirar");
        }
        try {
            System.out.println("Ingrés al compte");
            myAccount.depositAmount(1695);
        } catch (Exception e) {
            System.err.println(e.getMessage());
            System.out.println("Error en l'ingrés");
        }
        System.out.println("El saldo actual es " + myAccount.getBalance());
    }
}

```

RESULTADO:



```
Run main x
/home/xtrippsk8/GitHub/entornospractica/out/production/proyecto:/h
-console-standalone-1.10.0.jar SimpleBankingSystem.main
No hi ha suficient saldo
Error al retirar
Ingrés al compte
El saldo actual es 3495.0
Process finished with exit code 0
```

Part 2: Control de versions amb GitHub

1. Crear un repositori GitHub: Heu de crear un repositori a GitHub per gestionar el codi.
2. Crear branques:
 - a. Branca 1: Proves unitàries del dipòsit (tests-deposit).
`git checkout -b tests-deposit`
 - b. Branca 2: Proves unitàries del retir (tests-withdraw).
 - c. Branca 3: Refactorització (refactor-main).
3. Fer commits separats per cada test.

Part 3: Proves unitàries

Objectiu: Escriure proves unitàries per als mètodes `depositAmount()` i `withdrawAmount()` per garantir que el codi funciona correctament i gestionar les excepcions adequadament.

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.

```

1 package SimpleBankingSystem;
2
3 import org.junit.jupiter.api.Test;
4 import static org.junit.jupiter.api.Assertions.*;
5
6
7 public class AccountTest {
8
9     @Test
10    public void testDepositAmount() {
11        Account account = new Account("Flor Martinez", "1000-1234-56-123456789", 2500);
12
13        try {
14            account.depositAmount(500);
15            assertEquals(3000, account.getBalance(), "El saldo hauria de ser 3000 després del dipòsit.");
16        } catch (Exception e) {
17            fail("No hauria d'haver llançat cap excepció per un dipòsit vàlid.");
18        }
19
20        Exception exception = assertThrows(Exception.class, () -> {
21            account.depositAmount(-100);
22        });
23        assertEquals("No es pot ingressar una quantitat negativa.", exception.getMessage());
24    }
25
26    @Test
27    public void testWithdrawAmount() {
28        // Crear un compte amb saldo inicial
29        Account account = new Account("Flor Martinez", "1000-1234-56-123456789", 2500);
30
31        // Prova d'una retirada vàlida
32        try {
33            account.withdrawAmount(2000);
34            assertEquals(500, account.getBalance(), "El saldo hauria de ser 500 després de la retirada.");
35        } catch (Exception e) {
36            fail("No hauria d'haver llançat cap excepció per una retirada vàlida.");
37        }
38
39        // Prova d'una retirada superior al saldo
40        Exception exceptionInsufficientFunds = assertThrows(Exception.class, () -> {
41            account.withdrawAmount(3000);
42        });
43        assertEquals("No hi ha suficient saldo", exceptionInsufficientFunds.getMessage());
44
45        // Prova d'una retirada negativa
46        Exception exceptionNegativeAmount = assertThrows(Exception.class, () -> {
47            account.withdrawAmount(-100);
48        });
49        assertEquals("No es pot retirar una quantitat negativa.", exceptionNegativeAmount.getMessage());
50    }
51 }
52

```

2. Les proves han de verificar casos d'èxit, així com casos amb errors (quantitat negativa, saldo insuficient). Utilitzeu el mètode `assertEquals`.

CASOS DE EXITOS :

```
Run AccountTest x
✓ AccountTest (Sin 24 ms) ✓ Tests passed: 2 of 2 tests - 24 ms
/usr/lib/jvm/java-17-openjdk/bin/java ...
Process finished with exit code 0
```

CASOS DE FALLO :

```
32 try {
33     account.withdrawAmount(3000);
34     assertEquals(expected: 500, account.getBalance(), message: "El saldo hauria de ser 500 després de la retirada.");
35 } catch (Exception e) {
36     fail("No hauria d'haver llançat cap excepció per una retirada vàlida.");
37 }
38
39 // Prova d'una retirada superior al saldo
40 Exception exceptionInsufficientFunds = assertThrows(Exception.class, () -> {
41     account.withdrawAmount(3000);
42 });
43 assertEquals(expected: "No hi ha suficient saldo", exceptionInsufficientFunds.getMessage());
44
45 // Prova d'una retirada negativa
Run AccountTest x
✗ AccountTest (Sin 32 ms) ✗ Tests failed: 1, passed: 1 of 2 tests - 32 ms
✗ testWithdrawAr 6 ms
/usr/lib/jvm/java-17-openjdk/bin/java ...
org.opentest4j.AssertionFailedError: No hauria d'haver llançat cap excepció per una retirada vàlida.
```

Part 4: Refactorització del codi

La refactorització ha de seguir el patró Extract Method, que consisteix en separar les responsabilitats dins de la classe Main en mètodes independents, seguint les pràctiques de refactorització descrites a la pàgina web de Refactoring Guru <https://refactoring.guru/extract-method>.

En la classe Main, actualment tenim un mètode main que té moltes responsabilitats: gestionar el compte, realitzar un ingrés, retirar diners i manejar les excepcions. Segons les pràctiques de refactorització, podem dividir aquesta responsabilitat en diversos mètodes, per tal de simplificar el flux de l'aplicació.

Explica quines decisions i passos segueixes per refactoritzar el codi i el perquè.

Annex 1: Instruccions per modificar l'arxiu README:

Descarregar Markdown editor. Seguiu les instruccions del següent enllaç:
<https://davidrengifo.wordpress.com/2015/06/05/markdown-editor-plugin-for-eclipse-ide/>

Annex 2: Comandes git:

git status → Mostra l'estat del repositori, incloent canvis no compromesos, arxius nous...
git add <arxiu> → Afegeix arxius a l'àrea de preparació (staging area) per al commit.
git commit -m "<missatge>" → Realitza un commit amb els canvis que estan a l'àrea de preparació.

git push → Puja els canvis locals al repositori remot (per exemple, a GitHub, GitLab, etc.).

git pull → Baixa i fusiona els canvis del repositori remot al repositori local.

git branch → Mostra les branques locals del repositori.

git checkout <branca> → Canvia a la branca especificada.

git checkout -b <branca> → Crea i canvia a una nova branca en un sol pas.

git merge <branca> → Fusiona la branca especificada amb la branca actual.

git branch -a → Mostra totes les branques, tant locals com remotes.