

PART 1

Descripció inicial del codi:

- Què fa el codi? (Explicar breument).

El código simula un sistema bancario básico que permite realizar operaciones como depositar y retirar dinero de una cuenta. También maneja excepciones en caso de errores, como intentar retirar más dinero del disponible o ingresar cantidades negativas.

- Quins són els mètodes més importants i què fan?

`depositAmount(double amount)`: Añade una cantidad al saldo de la cuenta.

`withdrawAmount(double amount)`: Retira una cantidad del saldo de la cuenta.

`getBalance()`: Devuelve el saldo actual de la cuenta.

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

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?

Account.java

```
51         if (amount<0)
52             throw new Exception("No es pot ingressar una quantitat
53         balance += amount;
54     }
55
56     /* mètode per treure quantitats del compte
57     * modifica el saldo
58     */
59     public void withdrawAmount(double amount) throws Exception
60     {
61         if (amount < 0)
62             throw new Exception ("No es pot retirar una quantitat
63         if (getBalance()< amount)
64             throw new Exception ("No hi ha suficient saldo");
65         balance -= amount;
66     }
67
```

LINEA 51, 53, 61, 63

MAIN.java

```
• 10      myAccount = new Account("Flor Martinez", "1000-1234-56-1234");
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  }
```

Linea: 10,13,21,28

Estos puntos permiten observar cómo cambian las variables (name, account, balance) durante la ejecución del programa y cómo se manejan las excepciones.

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

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

```

4  * @author Flor Martinez
5  */
6  public class Main {
7      public static void main(String[] args) {
8          Account myAccount;
9
10         myAccount = new Account("Flor Martinez", "1000-1
11     }
  
```

Name	Value
no method return	
args	String[0] (id=20)

2.

```

12     try {
13         myAccount.withdrawAmount(2300);
14     } catch (Exception e) {
15         System.err.println(e.getMessage());
16         System.out.println("Error al retirar");
  
```

Name	Value
<init>() return	(No explicit return v.
args	String[0] (id=20)
myAccount	Account (id=27)

3.

```

28     return name;
29 }
30
31 //mètode per actualitzar el nom del titular del compte
32 public void setName(String name) {
33     this.name = name;
34 }
35
36 //mètode per tornar el numero de compte
37 public String getAccount() {
38     return account;
39 }
40
41 //mètode per tornar el saldo disponible del compte
42 public double getBalance() {
43     return balance;
44 }
45
46 /* mètode per ingressar quantitats al compte
47  * modifica el saldo
48  */
49 public void depositAmount(double amount) throws Exception {
50     {
51         if (amount < 0)
52             throw new Exception("No es pot ingressar una
53         balance += amount;
54     }
55 }
56
57 /* mètode per treure quantitats del compte
58  * modifica el saldo
59  */
60 public void withdrawAmount(double amount) throws Exception {
61     {
62         if (amount < 0)
  
```

Name	Value
no method return	
this	Account (id=27)
amount	2300.0

4.

The screenshot shows the IDE with `Account.java` open. A breakpoint is set at line 53. The Variables window on the right shows the state of the program:

Name	Value
no method return	
this	Account (id=27)
amount	2300.0

5.

The screenshot shows the IDE with `Account.java` open. A breakpoint is set at line 63. The Variables window on the right shows the state of the program:

Name	Value
getBalance() return	2500.0
this	Account (id=27)
amount	2300.0

6.

The screenshot shows a Java IDE with a code editor on the left and a console on the right. The code in the editor is as follows:

```

20 /**
21  *
22  * @author Flor Martinez
23  */
24 public class Main {
25     public static void main(String[] args) {
26         Account myAccount;
27
28         myAccount = new Account("Flor Martinez", "1000-1");
29
30         try {
31             myAccount.withdrawAmount(2300);
32         } catch (Exception e) {
33             System.err.println(e.getMessage());
34             System.out.println("Error al retirar");
35         }
36     }
37 }

```

The console on the right shows the following output:

Name	Value
withdrawAmount	(No explicit return)
args	String[0] (id=20)
myAccount	Account (id=27)

7.

The screenshot shows a Java IDE with a code editor on the left and a console on the right. The code in the editor is as follows:

```

1 package Pràctica_RA3_RA4;
2 /**
3  *
4  * @author Flor Martinez
5  */
6 public class Main {
7     public static void main(String[] args) {
8         Account myAccount;
9
10        myAccount = new Account("Flor Martinez", "1000-1");
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 }

```

The console on the right shows the following output:

Name	Value
println() returns	(No explicit return)
args	String[0] (id=20)
myAccount	Account (id=27)

8.

The screenshot shows a Java IDE with a code editor on the left and a console on the right. The code in the editor is as follows:

```

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

```

The console on the right shows the following output:

Name	Value
no method return	
this	Account (id=27)
amount	1695.0

9.

The screenshot shows a Java IDE with the following code in the editor:

```

28     return name;
29 }
30
31 //mètode per actualitzar el nom del titular del compte
32 public void setName(String name) {
33     this.name = name;
34 }
35
36 //mètode per tornar el numero de compte
37 public String getAccount() {
38     return account;
39 }
40
41 //mètode per tornar el saldo disponible del compte
42 public double getBalance() {
43     return balance;
44 }
45
46 /* mètode per ingressar quantitats al compte
47  * modifica el saldo
48  */
49 public void depositAmount(double amount) throws Exception
50 {
51     if (amount < 0)
52         throw new Exception("No es pot ingressar una
53         balance += amount;
54 }

```

On the right, the 'Variables' view shows the state of the object:

Name	Value
no method retu	
this	Account (id=27)
amount	1695.0

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.

cambio:

```

try {
    System.out.println("Ingrés al compte");
    myAccount.depositAmount(1695);
}

```

```

try {
    System.out.println("Ingrés al compte");
    myAccount.depositAmount(-1000);
}

```

```

<terminated> Main [Java Application] C:\Users\alumnat\p2\pool
Ingrés al compte
No es pot ingressar una quantitat negativa.
Error en l'ingrés
El saldo actual es 200.0

```

PART 3

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.

The screenshot shows an IDE with the following components:

- Left Panel (Run and Debug View):**
 - Top: "Finished after 0,02 seconds", "Runs: 5/5", "Errors: 0", "Failures: 0".
 - Tree view: "Pràctica_RA3_RA4.AccountTest [Runn...]"
 - testDepositAmountNegative (0,000 s)
 - testWithdrawAmountNegative (0,000 s)
 - testDepositAmount (0,000 s)
 - testWithdrawAmount (0,000 s)
 - testWithdrawAmountInsufficientB...
 - Bottom: "Failure Trace" (empty).
- Right Panel (Editor):**
 - File: "AccountTest.java"

```

1 package Pràctica_RA3_RA4;
2
3 import static org.junit.Assert.*;
4 import org.junit.Test;
5
6 public class AccountTest {
7
8     @Test
9     public void testDepositAmount() throws Exception {
10         // Caso de éxito: Depósito de una cantidad válida
11         Account account = new Account("Flor Martinez", "
12         account.depositAmount(500);
13         assertEquals(3000.0, account.getBalance(), 0.001
14     }
15
16     @Test(expected = Exception.class)
17     public void testDepositAmountNegative() throws Excep
18         // Caso de error: Depósito de una cantidad negat
19         Account account = new Account("Flor Martinez", "
20         account.depositAmount(-500); // Debe lanzar una
21     }
22
23     @Test
24     public void testWithdrawAmount() throws Exception {
25         // Caso de éxito: Retiro de una cantidad válida
26         Account account = new Account("Flor Martinez", "
27         account.withdrawAmount(500);
28         assertEquals(2000.0, account.getBalance(), 0.001
29     }
30
31     @Test(expected = Exception.class)
32     public void testWithdrawAmountNegative() throws Exce
33         // Caso de error: Retiro de una cantidad negativ
34         Account account = new Account("Flor Martinez", "
35         account.withdrawAmount(-500); // Debe lanzar una
36     }
37
38     @Test(expected = Exception.class)
39     public void testWithdrawAmountInsufficientBalance()
40         // Caso de error: Retiro de más dinero del dispo
41         Account account = new Account("Flor Martinez", "
42         account.withdrawAmount(3000); // Debe lanzar una

```
- Bottom Panel:** "Console", "Problems", "Debug Shell".

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