

INSTITUTO POLITÉCNICO NACIONAL



"ESCOM" (ESCUELA SUPERIOR DE CÓMPUTO)

DOCENTE: RODRIGUEZ CASTILLO MIGUEL ANGEL

INTEGRANTES:

VÁZQUEZ BLANCAS CÉSAR SAID

SEGUNDO CRUZ DANIEL

MENDOZA SEGURA FERNANDO

U.A.: PARADIGMAS DE PROGRAMACION

GRUPO: 3CM5

PRÁCTICA 4: CLASES E INSTANCIAS DE CLASE

---

# Planteamiento del problema

## Planteamiento del Problema 1

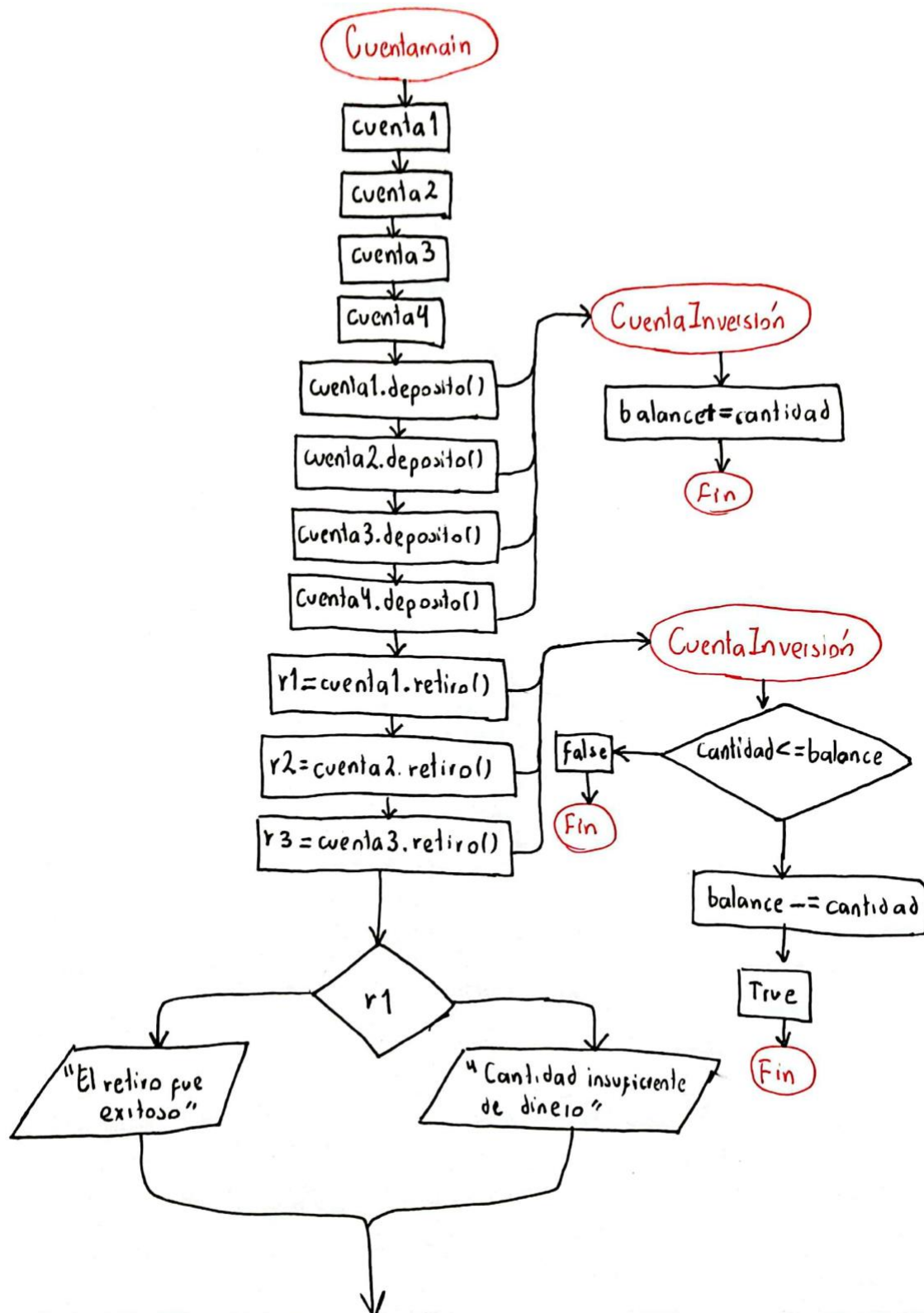
**Descripción del problema:** Aplicar los conocimientos adquiridos para el desarrollo de programas en Java en cuanto al concepto de Clases e instanciar las mismas, utilizando las buenas prácticas de programación en el diseño, implementación, pruebas y depuración del mismo.

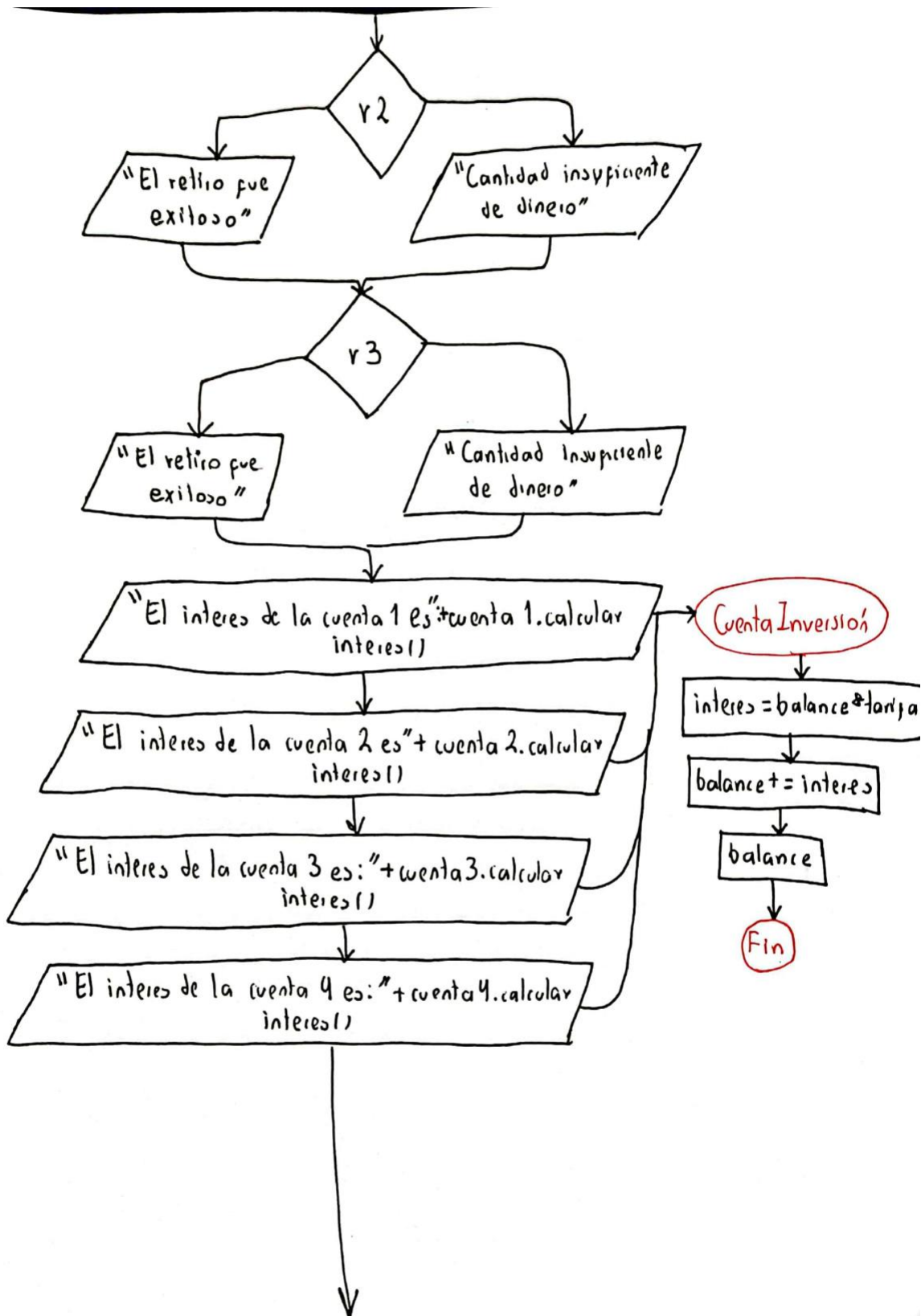
**Entrada:** Como entrada tenemos los datos dentro de la clase “Cuenta Inversión” los cuales son diferentes tipos de datos que representan las siguientes variables: un string para el “ID”, un double para el “balance”, otro double para la “tarifa” y finalmente un string para el “nombre”. De igual manera cabe mencionar que dichos datos son recibidos de forma mas especifica en el main para su procesamiento en los métodos y objetos que se declararon.

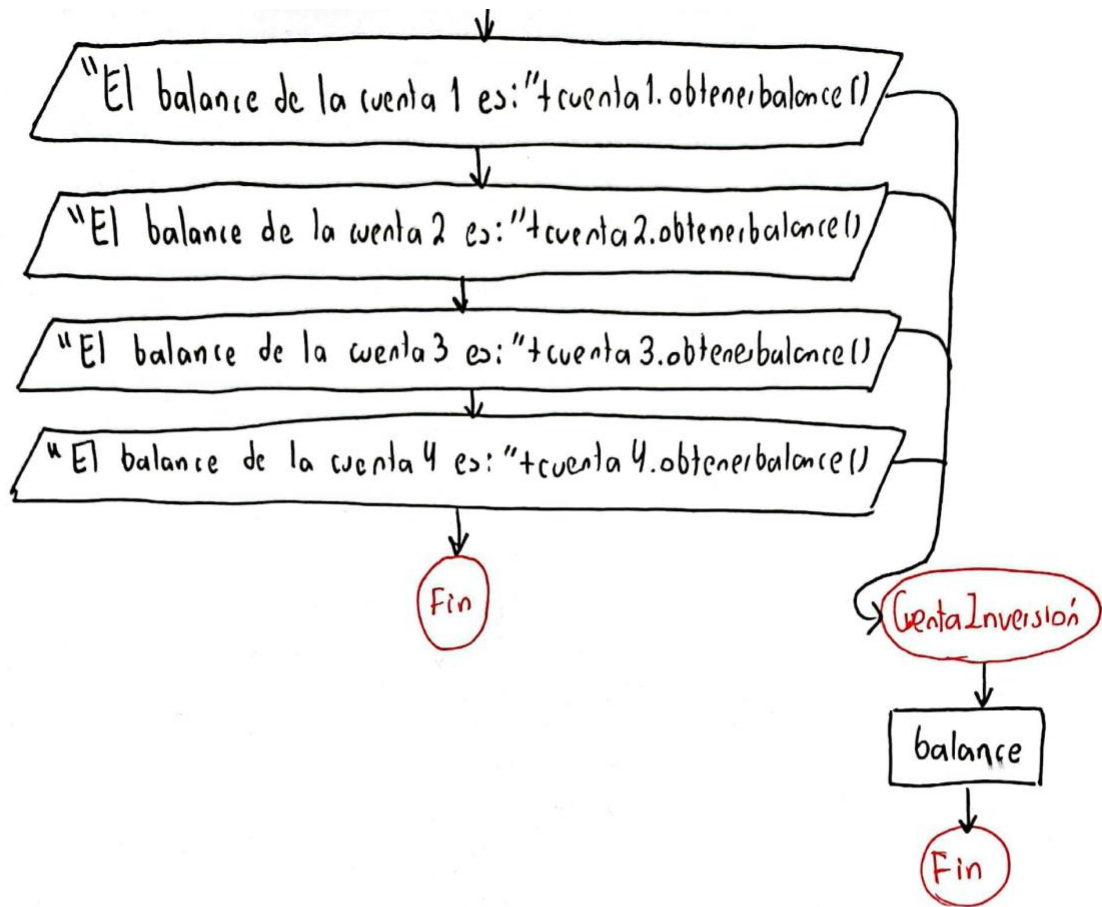
**Proceso:** En el procesamiento tenemos todo lo que es la implementación de los métodos y el uso de los constructores para el funcionamiento correcto del programa. Para la primera clase de cuenta inversión , el procesamiento es la programación de los constructores para los objetos dentro del main. Aquí es donde se declaran 4 constructores cada uno con diferente numero de parámetros, de igual manera los métodos de interés, balance, deposito y retiro son parte del proceso ya que utilizan los atributos para su funcionamiento. En cuanto a la clase “cuenta main” tenemos la creación de los objetos cuenta (4 cuentas) cada una con diferentes valores para sus respectivos parámetros, después tenemos la implementación de los métodos de depósito y retiro.

**Salida:** En cuanto a salida solamente tenemos los nuevos datos calculados de balance e interés de cada una de las cuentas mediante el uso de un `System.out.println()`.

# Diseño y funcionamiento de la solución







# Implementación de la solución

## Código desarrollado en lenguaje de programación



### CLASE “CuentaInversion”

```
src > packageBan > CuentaInversion.java > CuentaInversion > CuentaInversion(String, double, double)
1  package packageBan;
2
3  public class CuentaInversion {
4
5      private String Id;
6      private double balance;
7      private double tarifa;
8      private String nombre;
9
10     public CuentaInversion(String Id,double balance, double tarifa, String nombre){
11         this.tarifa=tarifa;
12         this.Id=Id;
13         this.balance=balance;
14         this.nombre=nombre;
15     }
16     public CuentaInversion(String Id,double balance, double tarifa){
17         this.tarifa=tarifa;
18         this.Id=Id;
19         this.balance=balance;
20     }
21     public CuentaInversion(String Id, double tarifa){
22         this.tarifa=tarifa;
23         this.Id=Id;
24     }
25     public CuentaInversion(double tarifa,double balance){
26         this.tarifa=tarifa;
27         this.balance=balance;
28     }
29     public double calcularinteres(){
30         double interes=balance*tarifa;
```

Ln 18, Col 24

```
27         this.balance=balance;
28     }
29     public double calcularinteres(){
30         double interes=balance*tarifa;
31         balance+=interes;
32         return interes;
33     }
34     public double obtenerbalance(){
35         return balance;
36     }
37     public void deposito(double cantidad){
38         balance+=cantidad;
39     }
40     public boolean retiro(double cantidad){
41         if(cantidad<=balance){
42             balance-=cantidad;
43             return true;
44         }
45         return false;
46     }
47
48 }
49
```

# CLASE "CuentaMain"

src > packageBan >  Cuentamain.java >  Cuentamain

```
1  package packageBan;
2
3  public class Cuentamain {
    Run | Debug
4      public static void main(String[] args) {
5          CuentaInversion cuenta1=new CuentaInversion(tarifa:0.05, balance:2500);
6          CuentaInversion cuenta2=new CuentaInversion(Id:"1", tarifa:0.07);
7          CuentaInversion cuenta3=new CuentaInversion(Id:"2", balance:2500.5, tarifa:0.07);
8          CuentaInversion cuenta4=new CuentaInversion(Id:"3", balance:4350.8, tarifa:0.11, nombre:"Samuel");
9          cuenta1.deposito(cantidad:7500.6);
10         cuenta2.deposito(cantidad:6600.4);
11         cuenta3.deposito(cantidad:3340.7);
12         cuenta4.deposito(cantidad:7789.6);
13         boolean r1=cuenta1.retiro(cantidad:2279);
14         boolean r2=cuenta3.retiro(cantidad:4579.8);
15         boolean r3=cuenta4.retiro(cantidad:12500);
16         if(r1){
17             System.out.println(x:"El retiro fue exitoso");
18         }else{
19             System.out.println(x:"Cantidad insuficiente de dinero");
20         }
21         if(r2){
22             System.out.println(x:"El retiro fue exitoso");
23         }else{
24             System.out.println(x:"Cantidad insuficiente de dinero");
25         }
26         if(r3){
27             System.out.println(x:"El retiro fue exitoso");
28         }else{
29             System.out.println(x:"Cantidad insuficiente de dinero");
```



```

25     }
26     if(r3){
27         System.out.println(x:"El retiro fue exitoso");
28     }else{
29         System.out.println(x:"Cantidad insuficiente de dinero");
30     }
31     System.out.println("El interes de la cuenta 1 es: "+cuenta1.calcularinteres());
32     System.out.println("El interes de la cuenta 2 es: "+cuenta2.calcularinteres());
33     System.out.println("El interes de la cuenta 3 es: "+cuenta3.calcularinteres());
34     System.out.println("El interes de la cuenta 4 es: "+cuenta4.calcularinteres());
35     System.out.println("El balance de la cuenta 1 es: "+cuenta1.obtenerbalance());
36     System.out.println("El balance de la cuenta 2 es: "+cuenta2.obtenerbalance());
37     System.out.println("El balance de la cuenta 3 es: "+cuenta3.obtenerbalance());
38     System.out.println("El balance de la cuenta 4 es: "+cuenta4.obtenerbalance());
39 }
40
41
42 }

```

El código no es difícil de explicar sin embargo requiere de un análisis detenido ya que se hace uso de los constructores y los métodos vistos en diagramas de clases.

Empezaremos explicando el funcionamiento de la clase “CuentaInversion”. De primera instancia tiene la declaración de los atributos en private los cuales son el ID, la tarifa, el balance y el nombre, cada dato con su respectivo tipo de variable.

Después tenemos la creación del primer constructor el cual es de tipo public y después viene el nombre de la clase constructor, por consiguiente, se colocan los paréntesis y dentro de ellos se colocan cada uno de los parámetros. Al hacer una sobrecarga de constructores ya que tendríamos mas de un constructor del mismo tipo, debemos a cada constructor diferente , colocarle un diferente numero o tipo de parámetros. Siguiendo el principio anterior, se crearon los 4 constructores diferentes que se solicitaron.

```

public CuentaInversion(String Id,double balance, double tarifa, String nombre){
    this.tarifa=tarifa;
    this.Id=Id;
    this.balance=balance;
    this.nombre=nombre;
}
public CuentaInversion(String Id,double balance, double tarifa){
    this.tarifa=tarifa;
    this.Id=Id;
    this.balance=balance;
}
public CuentaInversion(String Id, double tarifa){
    this.tarifa=tarifa;
    this.Id=Id;
}
public CuentaInversion(double tarifa,double balance){
    this.tarifa=tarifa;
    this.balance=balance;
}
}

```

Cabe mencionar que tanto los métodos como los constructores se volvieron públicos para su acceso en todo el programa. La sobrecarga en los constructores nos permite indicar al programa que hay diferentes formas de crear un mismo objeto dependiendo de los parámetros que se le manden.

Ahora para el desarrollo de los métodos tenemos como primer método el “public double calcularinteres()”. Dicho método no recibe parámetros y devuelve el interés calculado mediante la multiplicación de el balance y la tarifa actual, asimismo, incrementa el resultado del interés calculado al balance, es decir, balance+=interes.

El método obtenerbalance(), no recibe parámetros y solamente retorna el balance. El método “public void deposito” recibe como parámetro una cantidad (double) e incrementa la cantidad obtenida al balance, no devuelve ningún atributo.

Finalmente, el método “public boolean retiro()”, recibe como parámetro una cantidad y verifica si dicha cantidad es menor que el balance, si esto es cierto, se resta la cantidad al balance y retorna un valor true, en caso contrario únicamente regresa un valor false. Los métodos expresados en código se verían de la siguiente forma.

```

public double calcularinteres(){
    double interes=balance*tarifa;
    balance+=interes;
    return interes;
}
public double obtenerbalance(){
    return balance;
}
public void deposito(double cantidad){
    balance+=cantidad;
}
public boolean retiro(double cantidad){
    if(cantidad<=balance){
        balance-=cantidad;
        return true;
    }
    return false;
}

```

Ahora explicaremos la clase “Cuentamain”. Lo primero dentro de esta clase es crear nuestros 4 objetos e instanciarlos con diferentes valores.

Nota: La sobrecarga de métodos es la creación de varios métodos con el mismo nombre, pero con diferente lista de tipos de parámetros. Java utiliza el número y tipo de parámetros para seleccionar cuál definición de método ejecutar.

Después hacemos uso del método deposito accediendo al mismo mediante el punto es decir para depositar una cantidad en la cuenta 1 hacemos lo siguiente “cuenta1.deposito(cantidad: 7500.6);”. Después hacemos 3 retiros , en la cuenta1, cuenta3 y cuenta4, cabe mencionar que estos valores se guardarían en un booleano ya que el mismo método solo retorna valores de false o true, entonces ejemplificando el retiro de dinero en la cuenta1 tendríamos “boolean r1 = cuenta1.retiro(cantidad: 2279);” como ya sabemos se accede al método mediante el punto y se define la cantidad a retirar de la cuenta, cabe mencionar que para guardar el retorno declaramos la variable r1 como boolean para definir si el retiro fue true o false.

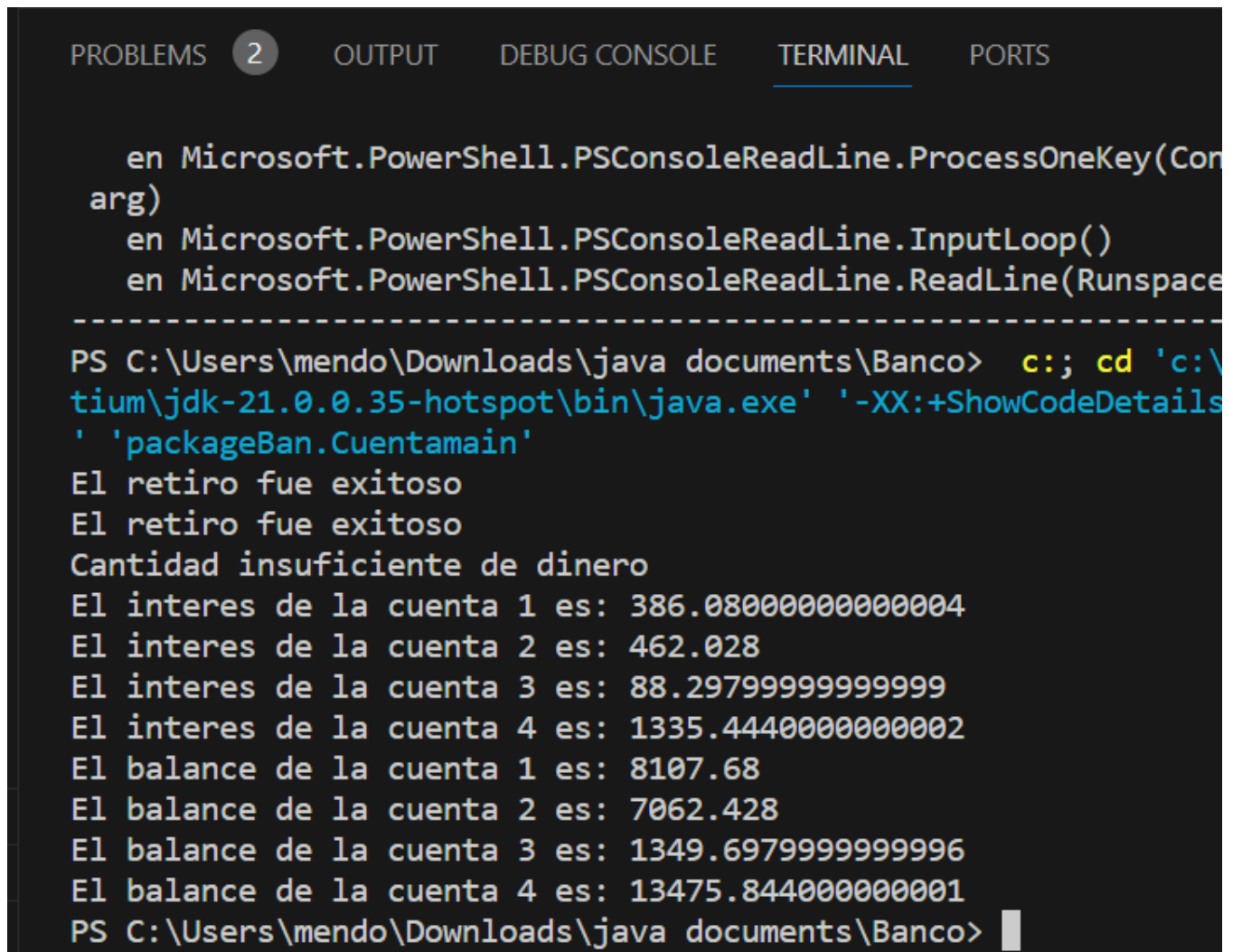
```
CuentaInversion cuenta1=new CuentaInversion(tarifa:0.05, balance:2500);
CuentaInversion cuenta2=new CuentaInversion(Id:"1", tarifa:0.07);
CuentaInversion cuenta3=new CuentaInversion(Id:"2", balance:2500.5, tarifa:0.07);
CuentaInversion cuenta4=new CuentaInversion(Id:"3", balance:4350.8, tarifa:0.11, nombre:"Samuel");
cuenta1.deposito(cantidad:7500.6);
cuenta2.deposito(cantidad:6600.4);
cuenta3.deposito(cantidad:3340.7);
cuenta4.deposito(cantidad:7789.6);
boolean r1=cuenta1.retiro(cantidad:2279);
boolean r2=cuenta3.retiro(cantidad:4579.8);
boolean r3=cuenta4.retiro(cantidad:12500);
if(r1){
    System.out.println(x:"El retiro fue exitoso");
}else{
    System.out.println(x:"Cantidad insuficiente de dinero");
}
if(r2){
    System.out.println(x:"El retiro fue exitoso");
}else{
    System.out.println(x:"Cantidad insuficiente de dinero");
}
if(r3){
    System.out.println(x:"El retiro fue exitoso");
}else{
    System.out.println(x:"Cantidad insuficiente de dinero");
}
```

Finalmente mediante condicionales determinamos si los 3 retiros de dinero fueron exitosos o no se pudieron debido una cantidad insuficiente en la cuenta e igualmente imprimimos en pantalla los valores de interés y balance de cada una de las 4 cuentas u objetos creados, los valores deberían estar modificados ya que se alteraron las cuentas: cuenta1, cuenta3 y cuenta4.

```
System.out.println("El interes de la cuenta 1 es: "+cuenta1.calcularinteres());
System.out.println("El interes de la cuenta 2 es: "+cuenta2.calcularinteres());
System.out.println("El interes de la cuenta 3 es: "+cuenta3.calcularinteres());
System.out.println("El interes de la cuenta 4 es: "+cuenta4.calcularinteres());
System.out.println("El balance de la cuenta 1 es: "+cuenta1.obtenerbalance());
System.out.println("El balance de la cuenta 2 es: "+cuenta2.obtenerbalance());
System.out.println("El balance de la cuenta 3 es: "+cuenta3.obtenerbalance());
System.out.println("El balance de la cuenta 4 es: "+cuenta4.obtenerbalance());
}
```

# Funcionamiento

Prueba de ejecución, resultados de salida y/o capturas de pantalla.



The screenshot shows an IDE interface with a terminal window. The terminal tabs at the top are PROBLEMS, 2, OUTPUT, DEBUG CONSOLE, TERMINAL, and PORTS. The terminal content shows the execution of a Java program. It starts with a command prompt where the user runs a Java command to compile and run a program. The output of the program is displayed below the command, showing various financial calculations and status messages.

```
en Microsoft.PowerShell.PSConsoleReadLine.ProcessOneKey(Con
arg)
en Microsoft.PowerShell.PSConsoleReadLine.InputLoop()
en Microsoft.PowerShell.PSConsoleReadLine.ReadLine(Runspace
-----
PS C:\Users\mendo\Downloads\java documents\Banco> c:; cd 'c:\
tium\jdk-21.0.0.35-hotspot\bin\java.exe' '-XX:+ShowCodeDetails
' 'packageBan.Cuentamain'
El retiro fue exitoso
El retiro fue exitoso
Cantidad insuficiente de dinero
El interes de la cuenta 1 es: 386.08000000000004
El interes de la cuenta 2 es: 462.028
El interes de la cuenta 3 es: 88.29799999999999
El interes de la cuenta 4 es: 1335.4440000000002
El balance de la cuenta 1 es: 8107.68
El balance de la cuenta 2 es: 7062.428
El balance de la cuenta 3 es: 1349.6979999999996
El balance de la cuenta 4 es: 13475.844000000001
PS C:\Users\mendo\Downloads\java documents\Banco>
```

## Prueba de escritorio

Cuenta	Depósito	retiro	cantidad ≤ balance	balance	interes	tar
1	—	—		2500	—	0.05
2	—	—		—	—	0.07
3	—	—		2500.5	—	0.07
4	—	—		4350.8	—	0.11
5	—	—		—	—	—
1	7500.6	—		10,000.6	—	0.05
2	6600.4	—		6600.4	—	0.07
3	3340.7	—		5841.2	—	0.07
4	7789.6	—		12,140.4	—	0.11
1	—	2279 ✓		7,721.6	—	0.05
2	—	—		6600.4	—	0.07
3	—	4579.8 ✓		1261.4	—	0.07
4	—	X		12,140.4	—	0.11
1	—	—	8107.68	386.8	0.05	
2	—	—	7062.428	462.028	0.07	
3	—	—	1349.698	88.298	0.07	
4	—	—	13475.844	1335.44	0.11	

# Conclusiones Individuales

## Vázquez Blancas Cesar Said

En esta practica se pudo ver cómo actúan la sobrecarga de constructores, metiendo 4 constructores así como la creación de objetos, enviando o no los parámetros y ver cómo estos pueden tener varios valores gracias a los atributos, la creación de objetos y su inicialización como variable, la creación de métodos y como aplicarlos a los objetos, uno por uno, con la idea de que cada objeto tiene sus propios valores individuales, y que en la aplicación, debes de hacer que cada atributo sea inicializado por su constructor correspondiente, en los métodos se hacen procesos y estructuras de control para que el funcionamiento sea el óptimo y eficaz, en esta practica se ve como cada objeto se va por su lado en el sentido de los datos, y esto hace que sea muy aplicable a nivel más del campo laboral, pues con los constructores se hacen cálculos simples pero que se mantendrán y se modificarán para simular un banco, el retorno y recibimiento de variables hace que el usuario pueda tener control de lo que se puede considerar su cuenta, dando lugar a una simulación básica pero funcional del banco y como sacar su interés, así como el hecho de mostrar datos e imprimirlos para ver cuáles son los datos finales y modificados a lo largo del programa de los objetos involucrados o de un solo objeto en cuestión

## Segundo Cruz Daniel

Esta práctica de programación orientada a objetos me permitió sumergirme en un enfoque de codificación diferente. En lugar de simplemente escribir líneas de código, comencé a pensar en términos de clases y objetos, lo cual fue un cambio de perspectiva interesante.

El uso de múltiples constructores en la clase `CuentaInversion` resultó ser una herramienta poderosa para personalizar la creación de instancias según mis necesidades. Me di cuenta de que la flexibilidad en la inicialización de objetos es una de las ventajas clave de la programación orientada a objetos. La noción de encapsulamiento, al ocultar los detalles internos de la implementación de la clase, se sintió como un paso importante hacia un código más organizado y mantenible. El hecho de que los métodos fueran específicos y claros en su propósito hizo que entender y utilizar la clase fuera más intuitivo. El manejo de errores en el método `retiro` destacó la importancia de anticipar posibles problemas y abordarlos de manera proactiva. Esta práctica me hizo apreciar la robustez que se puede lograr al diseñar clases de manera cuidadosa. La capacidad de un objeto para realizar cálculos internos y actualizar su propio estado, como se ve en el método `calcularInteres`, fue reveladora. Comencé a ver los objetos no solo como contenedores de datos, sino como entidades activas capaces de realizar acciones.

En `CuentaMain`, la interacción entre objetos se sintió natural. Crear instancias de la clase `CuentaInversion` y observar cómo interactuaban entre sí en un flujo lógico proporcionó una visión práctica de cómo las clases pueden colaborar para lograr un objetivo común.

## Mendoza Segura Fernando

En la elaboración de esta practica se pusieron en practica el uso de constructores y la sobrecarga de constructores para la creación de múltiples objetos. También se hizo uso de los métodos y el acceso a los mismos para la modificación de algunos atributos en diferentes cuentas anteriormente creadas e instanciadas con algunos valores. De igual manera me pareció muy interesante el uso de los métodos y si acceso a ellos para la modificación de algunos valores ya que muchas cosas se simplifican en programación funcional y orientada a objetos a diferencia de la estructurada que se vuelve más estricta en cosas como el paso de parámetros o el acceso a funciones y su uso en otros ámbitos del código.

De primera mano como siempre se creó un proyecto para que todas las clases estuvieran dentro de un package y de esta manera se pudiera tener acceso a valores en cada clase mediante métodos de acceso o declaraciones de tipo public como los métodos o constructores para que cada clase tenga acceso o de tipo private para que sean exclusivas de dicha clase como por ejemplo los atributos de la clase "CuentaInversion".

## Bibliografía:

IV – Sobrecarga de métodos y constructores, Ing. Carlos Alberto Román Zamitiz, Facultad de Ingeniería, UNAM, Recuperado el 14 de noviembre del 2023, de

[http://profesores.fi-b.unam.mx/carlos/java/java\\_basico4\\_6.html#:~:text=La%20sobrecarga%20de%20m%C3%A9todos%20es,cu%C3%A1l%20definici%C3%B3n%20de%20m%C3%A9todo%20ejecutar.](http://profesores.fi-b.unam.mx/carlos/java/java_basico4_6.html#:~:text=La%20sobrecarga%20de%20m%C3%A9todos%20es,cu%C3%A1l%20definici%C3%B3n%20de%20m%C3%A9todo%20ejecutar.)

Videotutorial Sobrecarga del constructor: destructor. - C# - LinkedIn, linkedin.com, Recuperado el 14 de noviembre del 2023, de

<https://es.linkedin.com/learning/c-sharp-programacion-orientada-a-objetos/sobrecarga-del-constructor-destructor#:~:text=La%20sobrecarga%20en%20los%20constructores,%22Estudiante%22%20que%20recib%CC%ADa%20par%C3%A1metros.>