



CICLO: [DAM]
MÓDULO DE [PROGRAMACIÓN]

[Tarea N° 07]

Alumno:
[Juan Carlos Filter Martín]
[15456141A]

Contenido

1. Documentos que se adjuntan a este informe.....	3
2. RA07_g) Se han realizado programas que implementen y utilicen jerarquías de clases.....	3
A) Crear proyecto en NetBeans denominado "CuentasBancarias"	3
B) Dentro de dicho proyecto, crear un paquete denominado "modeloBancario"	3
C) Dentro del paquete "modeloBancario", crear una clase denominada Cliente, que modele los distintos clientes del banco que tienen una cuenta asociada para almacenar su dinero.....	4
I. Las características clase Cliente.....	5
3. RA07_b) Se han utilizado modificadores para bloquear y forzar la herencia de clases y métodos.....	7
A) Dentro del paquete " <i>modeloBancario</i> ", crear una clase abstracta denominada "Cuenta", que modele las distintas cuentas bancarias que mantiene la entidad financiera donde los clientes depositan su dinero.....	7
I. Las características clase Cuenta.....	8
4. RA07_c) Se ha reconocido la incidencia de los constructores en la herencia.....	11
A) Clases padre y clases hijas, deben de tener implementados los constructores.....	11
5. RA07_d) Se han creado clases heredadas que sobrescriban la implementación de métodos de la superclase.....	12
A) Dentro del paquete "modeloBancario", crear una clase denominada "CuentaCorriente", que herede de la clase "Cuenta", que modele un tipo de cuenta con un interés fijo del 1.5%.....	12
I. Las características clase CuentaCorriente.....	13
Dentro del paquete "modeloBancario", crear una clase denominada "CuentaAhorro", que herede de la clase "Cuenta", que modele un tipo de cuenta con un interés variable y un saldo mínimo necesario.....	14
I. Las características clase CuentaAhorro.....	15
6. RA07_e) Se han diseñado y aplicado jerarquías de clases.....	17
A) Crear un entorno que importe el paquete "modeloBancario" y que permita probar las distintas clases Cuentas implementadas.....	17
B) Probar clases mediante la clase main "CuentasBancarias"	17
I. CuentaCorriente.....	17
II. CuentaAhorro.....	21
7. RA07_f) Se han probado y depurado las jerarquías de clases.....	25
Se han adjuntado captura sobre el funcionamiento de la aplicación y se han realizado comentarios en el código.....	25
Cliente.....	25
Cuenta (Abstract).....	26
CuentaCorriente (hereda de cuenta).....	27
CuentaAhorro (hereda de cuenta).....	28
CuentasBancarias (main class).....	29

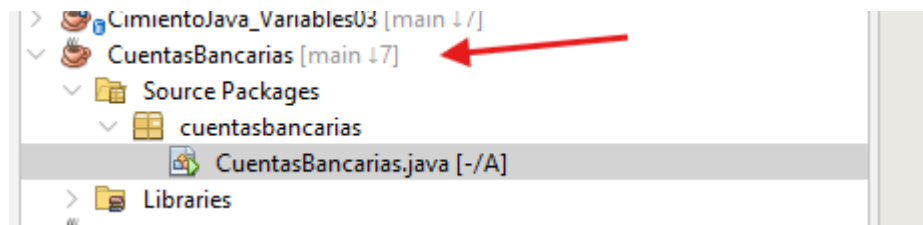
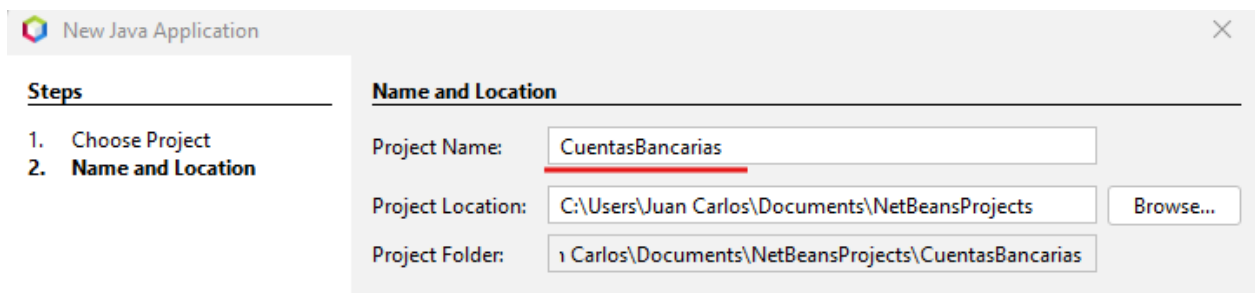
1. Documentos que se adjuntan a este informe.

A continuación se detallan los documentos que componen la presente entrega de la tarea:

1. Informe de elaboración de la tarea.
2. Proyecto java "CuentasBancarias".

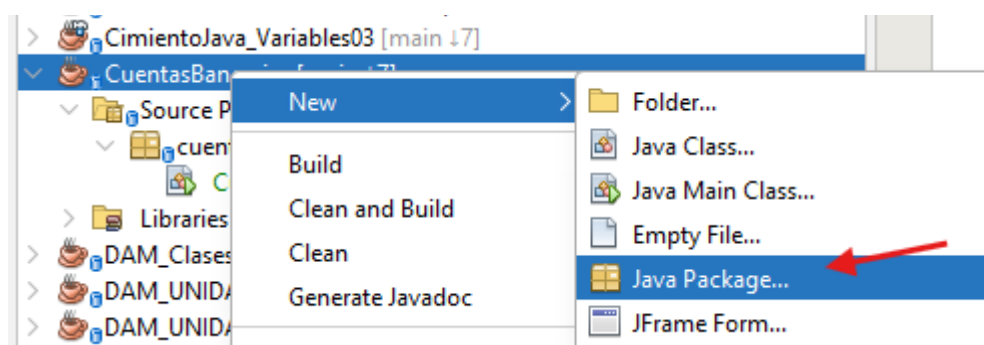
2. RA07_g) Se han realizado programas que implementen y utilicen jerarquías de clases.

A) Crear proyecto en NetBeans denominado "*CuentasBancarias*"

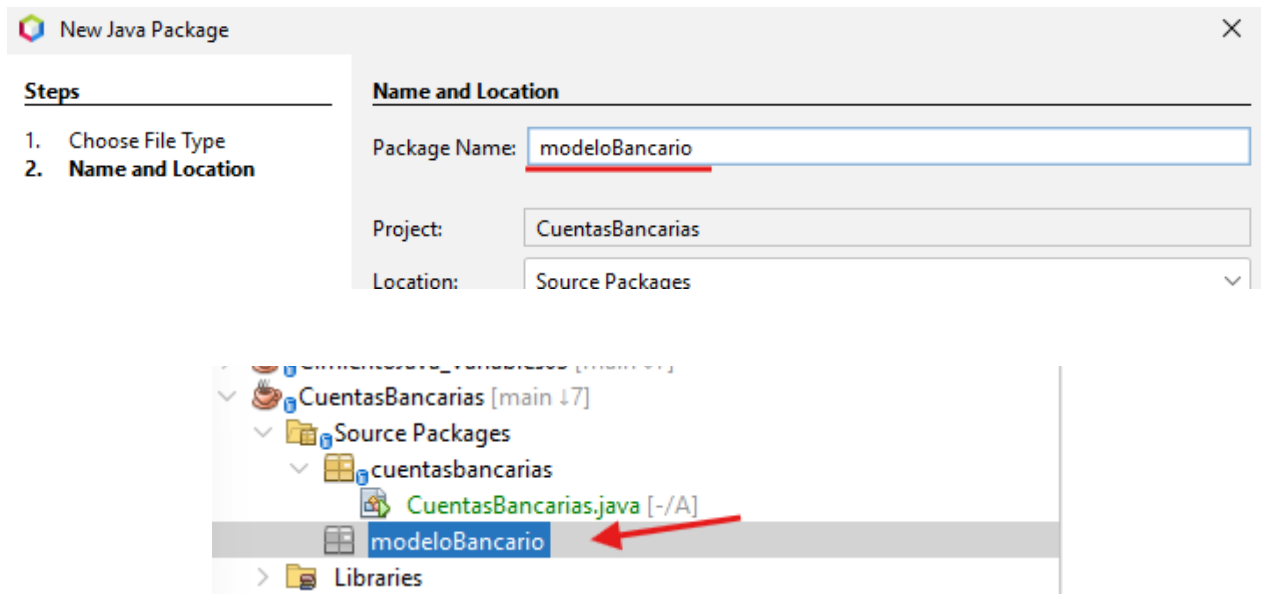


B) Dentro de dicho proyecto, crear un paquete denominado "*modeloBancario*"

Botón derecho sobre el proyecto > New > **Java Package**

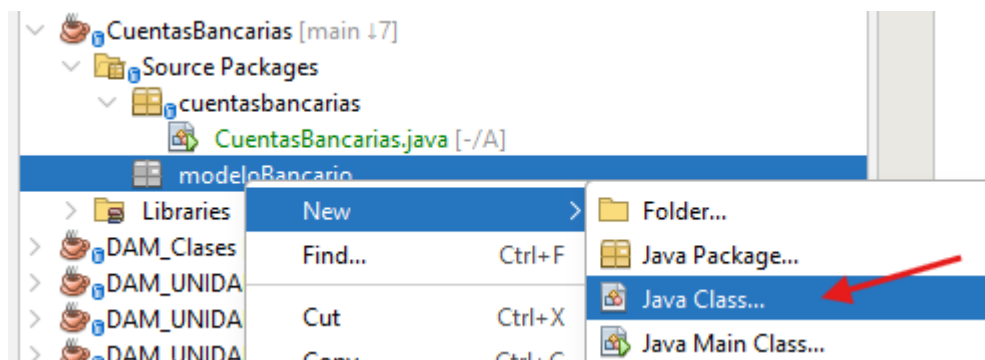


Le asignamos el nombre **modeloBancario**

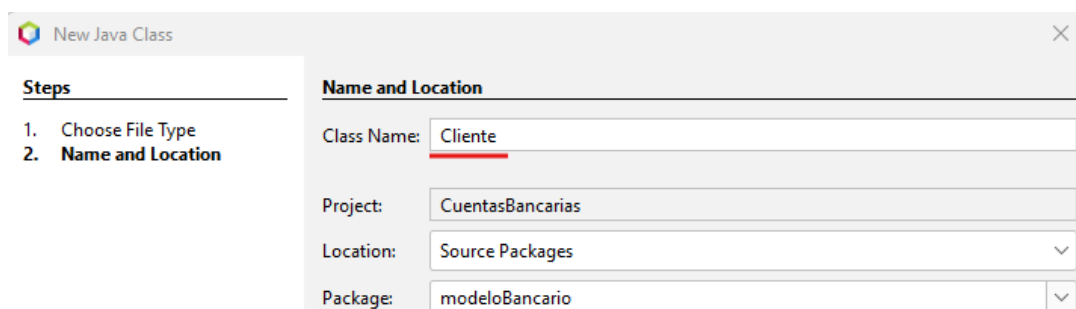


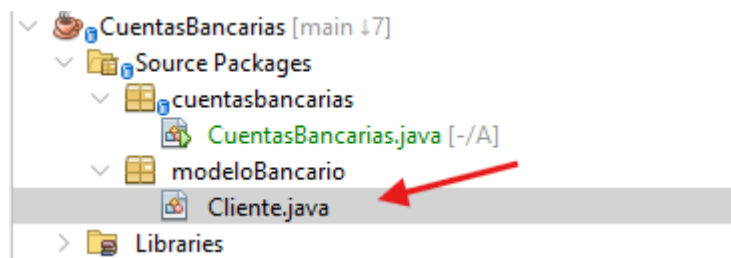
C) Dentro del paquete "*modeloBancario*", crear una clase denominada *Cliente*, que modele los distintos clientes del banco que tienen una cuenta asociada para almacenar su dinero.

Botón derecho sobre el paquete "modeloBancario" > New > **Java Class**



Le asignamos el nombre **Cliente** (dentro del paquete *modeloBancario*)





I. Las características clase Cliente

Atributos: *Los atributos con visibilidad privada.*

private: Para que sean con visibilidad privada se debe indicar mediante **private**

- **idCliente:** número entero que representa el identificador único del cliente dentro del banco.

private int idCliente

```
private int idCliente;
```

- **nombre:** cadena de caracteres que representa el nombre del cliente del banco.

private String nombre

```
private String nombre;
```

- **direccion:** cadena de caracteres que representa la dirección donde vive el cliente del banco.

private String direccion

```
private String direccion;
```

- **telefono:** cadena de caracteres que representa el teléfono que permite contactar con el cliente.

private String telefono

```
private String telefono;
```

Métodos Los métodos con visibilidad pública.

public: Para que sean con visibilidad pública se debe indicar mediante **public**

```
//Getters y Setters
public int getIdCliente() {
    return idCliente;
}

public void setIdCliente(int idCliente) {
    this.idCliente = idCliente;
}

public String getNombre() {
    return nombre;
}

public void setNombre(String nombre) {
    this.nombre = nombre;
}

public String getDireccion() {
    return direccion;
}

public void setDireccion(String direccion) {
    this.direccion = direccion;
}

public String getTelefono() {
    return telefono;
}

public void setTelefono(String telefono) {
    this.telefono = telefono;
}
```

Constructor sin parámetros: constructor que inicializa todos los atributos de tipo cadenas de caracteres al valor null y los números enteros a 0.

```
public Cliente() {  
    this.idCliente=0;  
    this.nombre=null;  
    this.direccion=null;  
    this.telefono=null;  
}
```

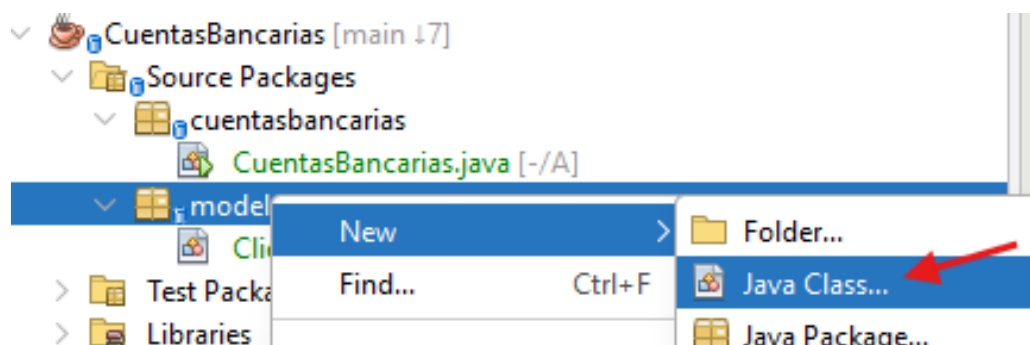
Constructor con parámetros: constructor que tienen tantos parámetros como atributos tiene la clase, y que inicializa cada uno de los atributos con el valor de los parámetros correspondientes.

```
public Cliente(int idCliente, String nombre, String direccion, String telefono) {  
    this.idCliente = idCliente;  
    this.nombre = nombre;  
    this.direccion = direccion;  
    this.telefono = telefono;  
}
```

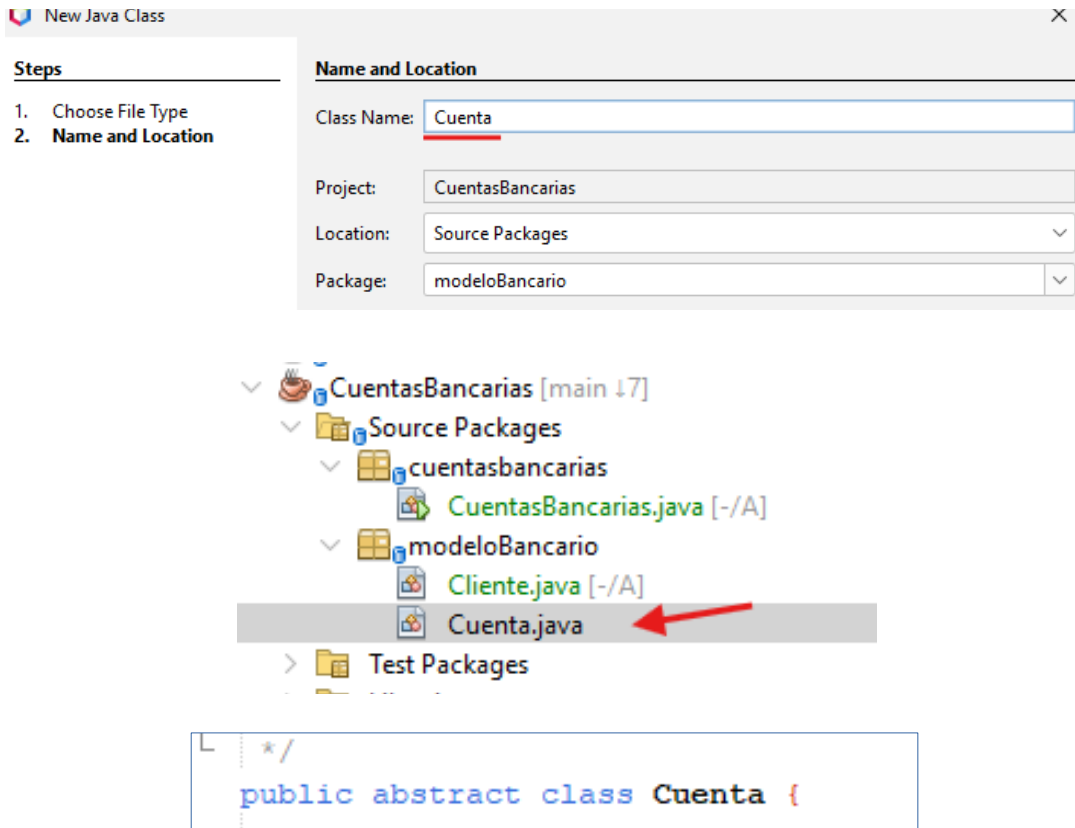
3. RA07_b) Se han utilizado modificadores para bloquear y forzar la herencia de clases y métodos.

A) Dentro del paquete "*modeloBancario*", crear una clase abstracta denominada "Cuenta", que modele las distintas cuentas bancarias que mantiene la entidad financiera donde los clientes depositan su dinero.

Botón derecho sobre el paquete "modeloBancario" > New > **Java Class**



Le asignamos el nombre **Cuenta** (dentro del paquete `modeloBancario`)



I. Las características clase Cuenta

Atributos: *Los atributos con visibilidad protegida.*

protected : Para que sean con visibilidad protegida se debe indicar mediante **protected**.

- numeroDeCuenta:** número entero que representa el identificador único asociado a cada una de las cuentas del banco.

protected `int` numeroDeCuenta

```
protected int numeroDeCuenta;
```

- saldo:** número real que representa la cantidad de dinero almacenado en dicha cuenta.

protected `double` saldo

```
protected double saldo;
```

- titular:** atributo de tipo Cliente que representa la persona que está asociada a dicha cuenta.

protected `Cliente` titular

```
protected Cliente titular;
```


Métodos Los métodos con visibilidad pública.

public: Para que sean con visibilidad pública se debe indicar mediante **public**

El método getSaldo redondea el saldo a 2 decimales y devuelve saldo

```
//Getters y Setters
public int getNumeroDeCuenta() {
    return numeroDeCuenta;
}


public void setNumeroDeCuenta(int numeroDeCuenta) {
    this.numeroDeCuenta = numeroDeCuenta;
}

public double getSaldo() {
    this.saldo = Math.round(this.saldo * 100) / 100d;
    return saldo;
}

public void setSaldo(double saldo) {
    this.saldo = saldo;
}

public Cliente getTitular() {
    return titular;
}

public void setTitular(Cliente titular) {
    this.titular = titular;
}
```



ingresar: recibe un parámetro entero que representa la cantidad que se desea ingresar en la cuenta. El método incrementará el saldo en la cantidad recibida como parámetro.

```
//Metodo ingresar
public void ingresar(double ingresarSaldo) {
    this.saldo += ingresarSaldo; //Incrementa el saldo sumandolo al atributo saldo
    System.out.println("Se ha ingresado " + ingresarSaldo + " euros");
}
```

retirar: método abstracto que permitirá sacar una cantidad de la cuenta (si hay saldo disponible para ello), no se implementará ya que dependerá del tipo de cuenta, por tanto su implementación recaerá en las clases hijas.

```
abstract public void retirar(double retirarSaldo) throws Exception;
```

actualizarSaldo: método abstracto que actualizará el saldo de la cuenta, dependiendo del tipo de interés de cada una de las cuenta, por tanto su implementación recaerá en las clases hijas.

```
abstract public void actualizarSaldo();
```

Constructor sin parámetros: constructor que inicializa el cliente titular de la cuenta a null, y el saldo y el número de cuenta a cero.

```
public Cuenta() {
    this.titular = null;
    this.saldo = 0;
    this.numeroDeCuenta = 0;
}
```

Constructor con parámetros: constructor que tienen tantos parámetros como atributos tiene la clase, y que inicializa cada uno de los atributos con el valor de los parámetros correspondientes.

```
public Cuenta(int numeroDeCuenta, double saldo, Cliente titular) {
    this.numeroDeCuenta = numeroDeCuenta;
    this.saldo = saldo;
    this.titular = titular;
}
```

4. RA07_c) Se ha reconocido la incidencia de los constructores en la herencia.

A) Clases padre y clases hijas, deben de tener implementados los constructores.

→ Clase padre Cuenta (Abstracta)

```
public abstract class Cuenta {  
  
    protected int numeroDeCuenta;  
    protected float saldo;  
    protected Cliente titular;  
  
    public Cuenta() {  
        this.titular = null;  
        this.saldo = 0;  
        this.numeroDeCuenta = 0;  
    }  
  
    public Cuenta(int numeroDeCuenta, float saldo, Cliente titular) {  
        this.numeroDeCuenta = numeroDeCuenta;  
        this.saldo = saldo;  
        this.titular = titular;  
    }  
}
```

Constructor sin parámetros

Constructor con parámetros

→ Clase hija CuentaCorriente

Constructor con parámetros

```
public class CuentaCorriente extends Cuenta {  
  
    protected final static double INTERES_FIJO = 0.015;  
  
    public CuentaCorriente(int numeroDeCuenta, float saldo, Cliente titular) {  
        super(numeroDeCuenta, saldo, titular);  
    }  
}
```

→ Clase hija CuentaAhorro

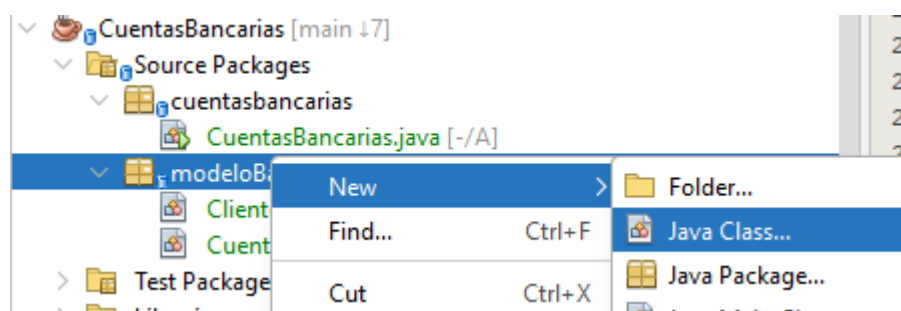
Constructor con parámetros

```
public class CuentaAhorro extends Cuenta {  
  
    protected double interesVariable;  
    protected double saldoMinimo;  
  
    public CuentaAhorro(int numeroDeCuenta, float saldo, Cliente titular,  
        double interesVariable, double saldoMinimo) {  
        super(numeroDeCuenta, saldo, titular);  
        this.interesVariable = interesVariable;  
        this.saldoMinimo = saldoMinimo;  
    }  
}
```

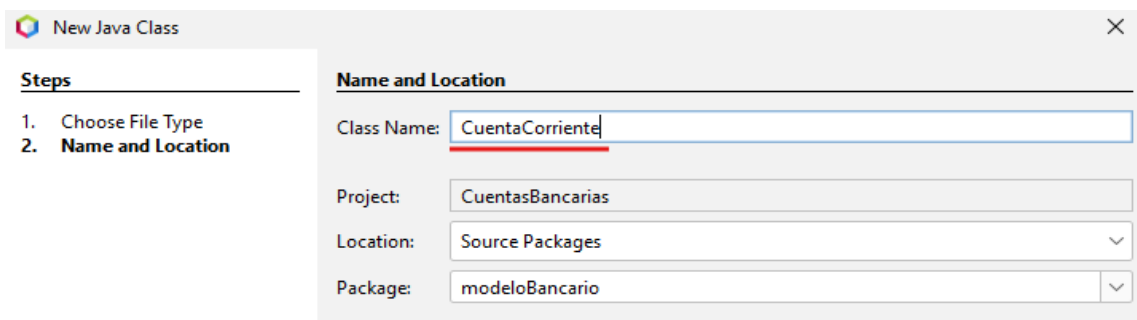
5. RA07_d) Se han creado clases heredadas que sobrescriban la implementación de métodos de la superclase.

A) Dentro del paquete "modeloBancario", crear una clase denominada "CuentaCorriente", que herede de la clase "Cuenta", que modele un tipo de cuenta con un interés fijo del 1.5%.

Botón derecho sobre el proyecto > New > **Java Class**



Le asignamos el nombre **CuentaCorriente** (dentro del paquete *modeloBancario*)



Hereda de la clase Cuenta

`public class CuentaCorriente extends Cuenta`

```
public class CuentaCorriente extends Cuenta{
```

I. Las características clase CuentaCorriente

Atributos Los atributos de la clase deben tener visibilidad protegida.

protected : Para que sean con visibilidad protegida se debe indicar mediante ***protected***.

interesFijo: constante real cuyo valor es 0.015.

```
protected final static double INTERES_FIJO = 0.015;
```

Métodos Los métodos deben tener visibilidad pública.

Métodos abstracto retirar:

```
@Override
public void retirar(double retirarSaldo) throws Exception {
1  if (retirarSaldo <= 0) {
    throw new Exception(message:"La cantidad que desea retirar tiene que ser positiva");
2  } else if (this.saldo < retirarSaldo) {
    throw new Exception(message:"Saldo insuficiente");
3  } else {
    this.saldo -= retirarSaldo;
    System.out.println("Se ha retirado " + retirarSaldo + " euros");
  }
}
```

1 – Si retirarSaldo(indicado por parámetros) es 0 o <: *El saldo a retirar debe ser positivo.*

2 – Si saldo es menor a retirarSaldo: *Insuficiente saldo.*

3 – Si no se cumple lo anterior entonces resta retirarSaldo al saldo de la cuenta.

Métodos abstracto actualizarSaldo.

```
@Override
public void actualizarSaldo() {
    this.saldo += this.saldo * INTERES_FIJO;

    //multiplicamos por 100 para obtener el porcentaje de 1.5% en vez de 0.015
    double Porcentaje= INTERES_FIJO * 100;
    System.out.println("El saldo ha sido actualizado con un interes del " + Porcentaje + "%");
}
```

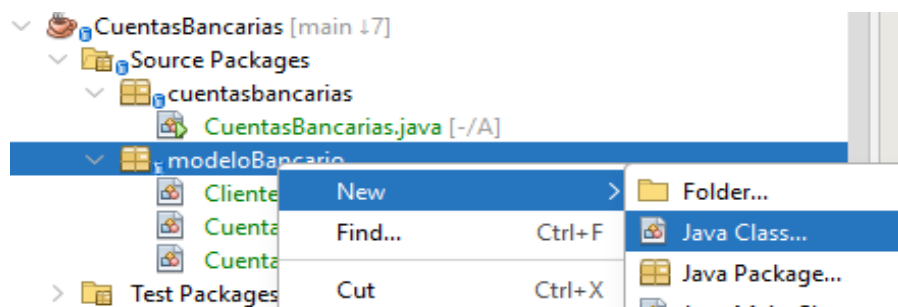
1 – Actualizamos el saldo de la cuenta añadiéndole el interés fijo.

Constructor con parámetros: constructor que tienen tantos parámetros como atributos tiene la clase, y que inicializa cada uno de los atributos con el valor de los parámetros correspondientes. Dicho constructor debe hacer uso del constructor de la clase padre "Cuenta".

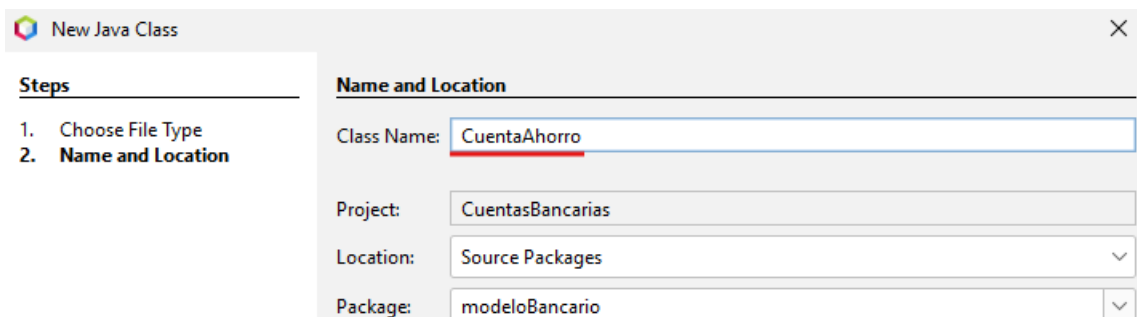
```
//Constructor con parametros
public CuentaCorriente(int numeroDeCuenta, double saldo, Cliente titular) {
    super(numeroDeCuenta, saldo, titular);
}
```

Dentro del paquete "modeloBancario", crear una clase denominada "CuentaAhorro", que herede de la clase "Cuenta", que modele un tipo de cuenta con un interés variable y un saldo mínimo necesario.

Botón derecho sobre el proyecto > New > Java Class



Le asignamos el nombre **CuentaAhorro** (dentro del paquete *modeloBancario*)



Hereda de la clase Cuenta

```
public class CuentaAhorro extends Cuenta
```

```
public class CuentaCorriente extends Cuenta{
```

I. Las características clase CuentaAhorro

Atributos Los atributos deben tener visibilidad protegida

protected : Para que sean con visibilidad protegida se debe indicar mediante ***protected***.

interesVariable: número real que representa el tipo de interés que se aplica a la cuenta.

```
protected double interesVariable;
```

saldoMinimo: número real que representa el dinero mínimo que debe haber en la cuenta.

```
protected double saldoMinimo;
```

Métodos (Todos los métodos de la clase CuentaAhorro deben tener visibilidad pública)

Métodos abstracto retirar:

```
@Override
public void retirar(double retirarSaldo) throws Exception{
1  if (retirarSaldo <= 0) {
2      throw new Exception(message:"La cantidad que desea retirar tiene que ser positiva");
3  } else if (this.saldo - retirarSaldo < saldoMinimo) {
      throw new Exception(message:"Saldo insuficiente. El saldo debe se mayor o igual al saldo Minimo");
  } else {
      this.saldo -= retirarSaldo;
      System.out.println("Se ha retirado " + retirarSaldo + " euros");
  }
}
```

1 – Si retirarSaldo(indicado por parámetros) es 0 o <: **El saldo a retirar debe ser positivo.**

2 – Si saldo se le resta retirarSaldo y es menor al saldoMinimo: **Insuficiente saldo**
(el saldosiempre debe ser mayor o igual al saldoMinimo)

3 – Si no se cumple lo anterior entonces resta retirarSaldo al saldo de la cuenta.

Métodos abstracto actualizarSaldo.

```
@Override
public void actualizarSaldo() {
1  double interes = this.interesVariable;
2  double aumentoSaldo = this.saldo / 5000 * 0.001; //Aumenta en 0.1% por cada 5000 euros de saldo
   interes += aumentoSaldo;
3  //Actualiza el saldo con el interes calculado
   this.saldo += this.saldo * interes;
4  //Desplazamos la coma 2 posiciones a la derecha
   double porcentaje = interes *100;
   //redondeamos a solamente 2 decimales
   porcentaje = Math.round(porcentaje *100)/100d;
5  System.out.println("El saldo ha sido actualizado con un interes del " + porcentaje + "%");
}
```

1 – Almacenamos el valor de interesVariable en interes

2 – Por cada 5000 aumentamos en 0.1% y almacenamos ese porcentaje en aumentoSaldo e indicamos que aumentoSaldo se suma al interes.

3 – Aplicamos el interes al saldo.

4 – multiplicamos por 100 el interes para obtener el % sin 0 a la izquierda y posteriormente redondeamos a 2 decimales.

5 – Mostramos por pantalla.

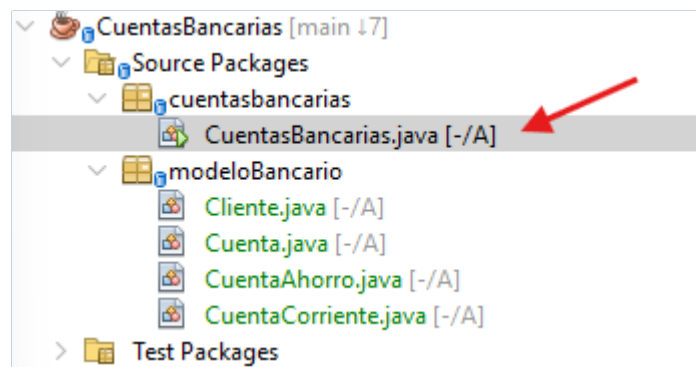
Constructor con parámetros: constructor que tienen tantos parámetros como atributos tiene la clase, y que inicializa cada uno de los atributos con el valor de los parámetros correspondientes. Dicho constructor debe hacer uso del constructor de la clase padre “Cuenta”.

```
//Constructor con parametros
public CuentaAhorro(int numeroDeCuenta, double saldo, Cliente titular,
    double interesVariable, double saldoMinimo) {
    super(numeroDeCuenta, saldo, titular);
    this.interesVariable = interesVariable;
    this.saldoMinimo = saldoMinimo;
}
```


6. RA07_e) Se han diseñado y aplicado jerarquías de clases.

A) Crear un entorno que importe el paquete *"modeloBancario"* y que permita probar las distintas clases Cuentas implementadas.

Esta clase está situada en el mismo proyecto pero en diferente paquete.



Entonces debemos importar el paquete modeloBancario a esta clase

```
package cuentasbancarias;
import modeloBancario.*;
/**
 *
 * @author Juan Carlos
 */
public class CuentasBancarias {
```

B) Probar clases mediante la clase main *"CuentasBancarias"*

I. CuentaCorriente

× Creamos un **objeto** de la **clase cliente** Con parámetros (*Registramos 1 cliente*)

```
public static void main(String[] args) {

    //Crear clientel
    Cliente clientel = new Cliente(idCliente: 1, nombre: "Juan Carlos",
    direccion: "Calle Arrecife 23", telefono: "699699699");
```

- × Creamos un **objeto** de la **clase CuentaCorriente**

```
//Crear cuenta corriente para clientel
CuentaCorriente cc = new CuentaCorriente(numeroDeCuenta: 1011, saldo: 22456.90, titular:clientel);
```

1 – Prueba con éxito

Mediante el objeto cuenta corriente “cc” le indicamos el método para ingresar, retirar y actualizar el saldo

Mostrando este mediante un System.out.println junto al método getSalgo que se encuentra en la clase cliente

```
//Realizar operaciones con los metodos
try {
    cc.ingresar(ingresarSaldo: 200); //Depositar
    cc.retirar(retirarSaldo: 500); //retirar
    cc.actualizarSaldo(); //Actualizar Saldo con interes fijo
    System.out.println("Saldo final en la cuenta Corriente: " + cc.getSaldo());
} catch (Exception e) {
    System.err.println(x: e.getMessage());
}
```

Resultado por pantalla:

```
public class CuentasBancarias {
    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {

        //Crear clientel
        Cliente clientel = new Cliente(idCliente: 1, nombre: "Juan Carlos",
            direccion: "Calle Arrecife 23", telefono: "699699699");

        //Crear CUENTA CORRIENTE para clientel
        CuentaCorriente cc = new CuentaCorriente(numeroDeCuenta: 1011, saldo: 22456.90,
            titular:clientel);

        //Realizar operaciones cuentaCorriente con los metodos
        try {
            cc.ingresar(ingresarSaldo: 200); //Depositar
            cc.retirar(retirarSaldo: 500); //retirar
            cc.actualizarSaldo(); //Actualizar Saldo con interes fijo
            System.out.println("Saldo final en la cuenta Corriente: " + cc.getSaldo());

        } catch (Exception e) {
            System.err.println(x: e.getMessage());
        }
    }
}
```

put ×

AM - C:\Users\Juan Carlos\Documents\JAVA\DAM × Debugger Console × CuentasBancarias (run) ×

```
run:
Se ha ingresado 200.0 euros
Se ha retirado 500.0 euros
El saldo ha sido actualizado con un interes del 1.5%
Saldo final en la cuenta Corriente: 22489.25
BUILD SUCCESSFUL (total time: 0 seconds)
```

2 – Prueba sin éxito

En el método retirar: al indicarle un valor de 0 o menor, mandará un mensaje de error

```
public void retirar(double retirarSaldo) throws Exception {
    if (retirarSaldo <= 0) {
        throw new Exception(message:"La cantidad que desea retirar tiene que ser positiva");
    } else if (this.saldo < retirarSaldo) {

//Realizar operaciones cuentaCorriente con los metodos
try {
    cc.ingresar(ingresarSaldo: 200); //Depositar
    cc.retirar(retirarSaldo: 0); //retirar
    cc.actualizarSaldo(); //Actualizar Saldo con interes fijo
    System.out.println("Saldo final en la cuenta Corriente: " + cc.getSaldo());
} catch (Exception e) {
    System.err.println(x: e.getMessage());
}
```

Resultado por pantalla:

```
public class CuentasBancarias {

/**
 * @param args the command line arguments
 */
public static void main(String[] args) {

    //Crear clientel
    Cliente clientel = new Cliente(idCliente: 1, nombre: "Juan Carlos",
        direccion: "Calle Arrecife 23", telefono: "699699699");

    //Crear CUENTA CORRIENTE para clientel
    CuentaCorriente cc = new CuentaCorriente(numeroDeCuenta: 1011, saldo: 22456.90,
        titular:clientel);

    //Realizar operaciones cuentaCorriente con los metodos
    try {
        cc.ingresar(ingresarSaldo: 200); //Depositar
        cc.retirar(retirarSaldo: 0); //retirar
        cc.actualizarSaldo(); //Actualizar Saldo con interes fijo
        System.out.println("Saldo final en la cuenta Corriente: " + cc.getSaldo());
    } catch (Exception e) {
        System.err.println(x: e.getMessage());
    }
}
```

out x

AM - C:\Users\Juan Carlos\Documents\JAVA\DAM x Debugger Console x CuentasBancarias (run) x

run:

Se ha ingresado 200.0 euros

La cantidad que desea retirar tiene que ser positiva

BUILD SUCCESSFUL (total time: 0 seconds)

3 – Prueba sin éxito

En el método `retirar`: al indicarle un valor mayor al saldo que tiene en la cuenta, mandará un mensaje de error

Se le ha indicado que el saldo en la cuenta sea de 500 y posteriormente se ha intentado retirar 1000

```
public void retirar(double retirarSaldo) throws Exception {  
    if (retirarSaldo <= 0) {  
        throw new Exception("La cantidad que desea retirar tiene que ser positiva");  
    } else if (this.saldo < retirarSaldo) {  
        throw new Exception("Saldo insuficiente");  
    } else {  
    }
```

```
//Crear CUENTA CORRIENTE para clientel  
CuentaCorriente cc = new CuentaCorriente(numeroDeCuenta: 1011, saldo: 500,  
    titular:clientel);  
  
//Realizar operaciones cuentaCorriente con los metodos  
try {  
    cc.ingresar(ingresarSaldo: 200); //Depositar  
    cc.retirar(retirarSaldo: 1000); //retirar  
    cc.actualizarSaldo(); //Actualizar Saldo con interes fijo  
    System.out.println("Saldo final en la cuenta Corriente: " + cc.getSaldo());  
} catch (Exception e) {  
    System.err.println(x: e.getMessage());  
}
```

Resultado por pantalla:

```
public static void main(String[] args) {  
  
    //Crear clientel  
    Cliente clientel = new Cliente(idCliente: 1, nombre: "Juan Carlos",  
        direccion: "Calle Arrecife 23", telefono: "699699699");  
  
    //Crear CUENTA CORRIENTE para clientel  
    CuentaCorriente cc = new CuentaCorriente(numeroDeCuenta: 1011, saldo: 500,  
        titular:clientel);  
  
    //Realizar operaciones cuentaCorriente con los metodos  
    try {  
        cc.ingresar(ingresarSaldo: 200); //Depositar  
        cc.retirar(retirarSaldo: 1000); //retirar  
        cc.actualizarSaldo(); //Actualizar Saldo con interes fijo  
        System.out.println("Saldo final en la cuenta Corriente: " + cc.getSaldo());  
    } catch (Exception e) {  
        System.err.println(x: e.getMessage());  
    }  
}
```

out x

AM - C:\Users\Juan Carlos\Documents\JAVA\DAM x Debugger Console x CuentasBancarias (run) x

run:

Se ha ingresado 200.0 euros

Saldo insuficiente

BUILD SUCCESSFUL (total time: 0 seconds)

II. CuentaAhorro

- × Creamos un **objeto** de la **clase cliente** Sin parámetros (*Registramos 1 cliente*)
- × Y asignamos los atributos mediante los Setters

```
//Crear cliente2
Cliente cliente2 = new Cliente();

cliente2.setIdCliente(idCliente: 2);
cliente2.setNombre(nombre: "Francisco");
cliente2.setDireccion(direccion: "Calle La Libertad 13");
cliente2.setTelefono(telefono: "688688688");
```

- × Creamos un **objeto** de la **clase CuentaAhorro**

```
//Crear cuenta corriente para cliente2
CuentaAhorro ca = new CuentaAhorro(numeroDeCuenta: 1022, saldo: 40221.43, titular: cliente2, interesVariable: 0.02, saldoMinimo: 20000);
```

1 – Prueba con éxito

Mediante el objeto cuenta ahorro “ca” le indicamos el método para ingresar, retirar y actualizar el saldo

Mostrando este mediante un System.out.println junto al método getSalgo que se encuentra en la clase cliente

```
//Realizar operaciones con los metodos
try {
    ca.ingresar(ingresarSaldo: 1500); //Depositar
    ca.retirar(retirarSaldo: 500); //retirar
    ca.actualizarSaldo(); //Actualizar Saldo con interes fijo
    System.out.println("Saldo final en la cuenta Ahorro: " + ca.getSaldo());
} catch (Exception e) {
    System.err.println(x: e.getMessage());
}
```

Resultado por pantalla:

```
public class CuentasBancarias {  
  
    /**  
     * @param args the command line arguments  
     */  
    public static void main(String[] args) {  
        //Crear cliente2  
        Cliente cliente2 = new Cliente();  
  
        cliente2.setIdCliente(idCliente: 2);  
        cliente2.setNombre(nombre: "Francisco");  
        cliente2.setDireccion(direccion: "Calle La Liberta 13");  
        cliente2.setTelefono(telefono: "688688688");  
  
        //Crear CUENTA AHORRO para cliente2  
        CuentaAhorro ca = new CuentaAhorro(numeroDeCuenta: 1022, saldo: 40221.43,  
            titular:cliente2, interesVariable:0.02, saldoMinimo:20000);  
  
        //Realizar operaciones cuentaAhorro con los metodos  
        try {  
            ca.ingresar(ingresarSaldo: 1500); //Depositar  
            ca.retirar(retirarSaldo: 500); //retirar  
            ca.actualizarSaldo(); //Actualizar Saldo con interes fijo  
            System.out.println("Saldo final en la cuenta Ahorro: " + ca.getSaldo());  
  
        } catch (Exception e) {  
            System.err.println(x: e.getMessage());  
        }  
    }  
}
```

out x

AM - C:\Users\Juan Carlos\Documents\JAVA\DAM x Debugger Console x CuentasBancarias (run) x

run:

```
Se ha ingresado 1500.0 euros  
Se ha retirado 500.0 euros  
El saldo ha sido actualizado con un interes del 2.82%  
Saldo final en la cuenta Ahorro: 42385.7  
BUILD SUCCESSFUL (total time: 0 seconds)
```

2 – Prueba sin éxito

En el método retirar: al indicarle un valor de 0 o menor, mandará un mensaje de error

```
public void retirar(double retirarSaldo) throws Exception{
    if (retirarSaldo <= 0) {
        throw new Exception(message: "La cantidad que desea retirar tiene que ser positiva");
    }

    //Realizar operaciones con los metodos
    try {
        ca.ingresar(ingresarSaldo: 600); //Depositar
        ca.retirar(retirarSaldo: -50); //retirar
        ca.actualizarSaldo(); //Actualizar Saldo con interes fijo
        System.out.println("Saldo final en la cuenta Ahorro: " + ca.getSaldo());
    } catch (Exception e) {
        System.err.println(x: e.getMessage());
    }
}
```

Resultado por pantalla:

```
public class CuentasBancarias {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        //Crear cliente2
        Cliente cliente2 = new Cliente();

        cliente2.setIdCliente(idCliente: 2);
        cliente2.setNombre(nombre: "Francisco");
        cliente2.setDireccion(direccion: "Calle La Liberta 13");
        cliente2.setTelefono(telefono: "688688688");

        //Crear CUENTA AHORRO para cliente2
        CuentaAhorro ca = new CuentaAhorro(numeroDeCuenta: 1022, saldo: 40221.43,
            titular: cliente2, interesVariable: 0.02, saldoMinimo: 20000);

        //Realizar operaciones cuentaAhorro con los metodos
        try {
            ca.ingresar(ingresarSaldo: 600); //Depositar
            ca.retirar(retirarSaldo: -50); //retirar
            ca.actualizarSaldo(); //Actualizar Saldo con interes fijo
            System.out.println("Saldo final en la cuenta Ahorro: " + ca.getSaldo());
        } catch (Exception e) {
            System.err.println(x: e.getMessage());
        }
    }
}
```

out x

AM - C:\Users\Juan Carlos\Documents\JAVA\DAM x Debugger Console x CuentasBancarias (run) x

run:

Se ha ingresado 600.0 euros

La cantidad que desea retirar tiene que ser positiva

BUILD SUCCESSFUL (total time: 0 seconds)

3 – Prueba sin éxito

En el método retirar: Si se le resta el valor de retirar saldo al saldo y este es menor o igual al saldo mínimo, mandará un mensaje de error

Se le ha indicado que el saldo en la cuenta sea de 22000 con un saldo mínimo de 20000 y posteriormente se ha intentado retirar 2300.

El dinero restante sería 19000 entonces mandará mensaje de error

```
//Crear CUENTA AHORRO para cliente2
CuentaAhorro ca = new CuentaAhorro(numeroDeCuenta: 1022, saldo: 22000,
    titular:cliente2, interesVariable:0.02, saldoMinimo:20000);

//Realizar operaciones cuentaAhorro con los metodos
try {
    ca.ingresar(ingresarSaldo: 0); //Depositar
    ca.retirar(retirarSaldo: 2300); //retirar
    ca.actualizarSaldo(); //Actualizar Saldo con interes fijo
    System.out.println("Saldo final en la cuenta Ahorro: " + ca.getSaldo());
} catch (Exception e) {
    System.err.println(x: e.getMessage());
}
```

Resultado por pantalla:

```
public static void main(String[] args) {
    //Crear cliente2
    Cliente cliente2 = new Cliente();

    cliente2.setIdCliente(idCliente: 2);
    cliente2.setNombre(nombre: "Francisco");
    cliente2.setDireccion(direccion: "Calle La Liberta 13");
    cliente2.setTelefono(telefono: "688688688");

    //Crear CUENTA AHORRO para cliente2
    CuentaAhorro ca = new CuentaAhorro(numeroDeCuenta: 1022, saldo: 22000,
        titular:cliente2, interesVariable:0.02, saldoMinimo:20000);

    //Realizar operaciones cuentaAhorro con los metodos
    try {
        ca.ingresar(ingresarSaldo: 0); //Depositar
        ca.retirar(retirarSaldo: 2300); //retirar
        ca.actualizarSaldo(); //Actualizar Saldo con interes fijo
        System.out.println("Saldo final en la cuenta Ahorro: " + ca.getSaldo());
    } catch (Exception e) {
        System.err.println(x: e.getMessage());
    }
}
```

put x

AM - C:\Users\Juan Carlos\Documents\JAVA\DAM x Debugger Console x CuentasBancarias (run) x

run:

Se ha ingresado 0.0 euros

Saldo insuficiente. El saldo debe se mayor o igual al saldo Minimo

BUILD SUCCESSFUL (total time: 0 seconds)

7. RA07_f) Se han probado y depurado las jerarquías de clases.

Se han adjuntado captura sobre el funcionamiento de la aplicación y se han realizado comentarios en el código.

Para finalizar se van a dejar capturas de cada clase de la aplicación

Cliente

```
public class Cliente {  
  
    //Atributos  
    private int idCliente;  
    private String nombre;  
    private String direccion;  
    private String telefono;  
  
    //Constructores  
    public Cliente() {  
        this.idCliente = 0;  
        this.nombre = null;  
        this.direccion = null;  
        this.telefono = null;  
    }  
  
    public Cliente(int idCliente, String nombre, String direccion, String telefono) {  
        this.idCliente = idCliente;  
        this.nombre = nombre;  
        this.direccion = direccion;  
        this.telefono = telefono;  
    }  
  
    //Getters y Setters  
    public int getIdCliente() {  
        return idCliente;  
    }  
  
    public void setIdCliente(int idCliente) {  
        this.idCliente = idCliente;  
    }  
  
    public String getNombre() {  
        return nombre;  
    }  
  
    public void setNombre(String nombre) {  
        this.nombre = nombre;  
    }  
  
    public String getDireccion() {  
        return direccion;  
    }  
  
    public void setDireccion(String direccion) {  
        this.direccion = direccion;  
    }  
  
    public String getTelefono() {  
        return telefono;  
    }  
  
    public void setTelefono(String telefono) {  
        this.telefono = telefono;  
    }  
}
```

Cuenta (Abstract)

```
public abstract class Cuenta {

    //Atributos
    protected int numeroDeCuenta;
    protected double saldo;
    protected Cliente titular;

    //Constructores
    public Cuenta() {
        this.titular = null;
        this.saldo = 0;
        this.numeroDeCuenta = 0;
    }

    public Cuenta(int numeroDeCuenta, double saldo, Cliente titular) {
        this.numeroDeCuenta = numeroDeCuenta;
        this.saldo = saldo;
        this.titular = titular;
    }

    //Getters y Setters
    public int getNumeroDeCuenta() {
        return numeroDeCuenta;
    }

    public void setNumeroDeCuenta(int numeroDeCuenta) {
        this.numeroDeCuenta = numeroDeCuenta;
    }

    public double getSaldo() {
        //redondea el saldo a dos decimales
        this.saldo = Math.round(this.saldo * 100) / 100d;
        return saldo;
    }

    public void setSaldo(double saldo) {
        this.saldo = saldo;
    }

    public Cliente getTitular() {
        return titular;
    }

    public void setTitular(Cliente titular) {
        this.titular = titular;
    }

    //Metodo ingresar
    public void ingresar(double ingresarSaldo) {
        this.saldo += ingresarSaldo; //Incrementa el saldo sumandolo al atributo saldo
        System.out.println("Se ha ingresado " + ingresarSaldo + " euros");
    }

    //metodos abstractos
    abstract public void retirar(double retirarSaldo) throws Exception;

    abstract public void actualizarSaldo();
}
```

CuentaCorriente (hereda de cuenta)

```
public class CuentaCorriente extends Cuenta {

    //Atributo constante
    protected final static double INTERES_FIJO = 0.015;

    //Constructor con parametros
    public CuentaCorriente(int numeroDeCuenta, double saldo, Cliente titular) {
        super(numeroDeCuenta, saldo, titular);
    }

    //metodos abstractos
    @Override
    public void retirar(double retirarSaldo) throws Exception {
        if (retirarSaldo <= 0) {
            throw new Exception(message:"La cantidad que desea retirar tiene que ser positiva");
        } else if (this.saldo < retirarSaldo) {
            throw new Exception(message:"Saldo insuficiente");
        } else {
            this.saldo -= retirarSaldo;
            System.out.println("Se ha retirado " + retirarSaldo + " euros");
        }
    }

    @Override
    public void actualizarSaldo() {
        this.saldo += this.saldo * INTERES_FIJO;

        //multiplicamos por 100 para obtener el porcentaje de 1.5% en vez de 0.015
        double Porcentaje= INTERES_FIJO * 100;
        System.out.println("El saldo ha sido actualizado con un interes del " + Porcentaje + "%");
    }
}
```

CuentaAhorro (hereda de cuenta)

```
public class CuentaAhorro extends Cuenta {

    //Atributos
    protected double interesVariable;
    protected double saldoMinimo;

    //Constructor con parametros
    public CuentaAhorro(int numeroDeCuenta, double saldo, Cliente titular,
        double interesVariable, double saldoMinimo) {
        super(numeroDeCuenta, saldo, titular);
        this.interesVariable = interesVariable;
        this.saldoMinimo = saldoMinimo;
    }

    //metodos abstractos
    @Override
    public void retirar(double retirarSaldo) throws Exception {
        if (retirarSaldo <= 0) {
            throw new Exception(message:"La cantidad que desea retirar tiene que ser positiva");
        } else if (this.saldo - retirarSaldo < saldoMinimo) {
            throw new Exception(message:"Saldo insuficiente. El saldo debe se mayor o igual al saldo Minimo");
        } else {
            this.saldo -= retirarSaldo;
            System.out.println("Se ha retirado " + retirarSaldo + " euros");
        }
    }

    @Override
    public void actualizarSaldo() {

        double interes = this.interesVariable;

        double aumentoSaldo = this.saldo / 5000 * 0.001; //Aumenta en 0.1% por cada 5000 euros de saldo
        interes += aumentoSaldo;

        //Actualiza el saldo con el interes calculado
        this.saldo += this.saldo * interes;

        //Desplazamos la coma 2 posiciones a la derecha
        double porcentaje = interes *100;
        //redondeamos a solamente 2 decimales
        porcentaje = Math.round(porcentaje *100)/100d;

        System.out.println("El saldo ha sido actualizado con un interes del " + porcentaje + "%");
    }
}
```

CuentasBancarias (main class)

```
package cuentasbancarias;

import modeloBancario.*;

/**
 *
 * @author Juan Carlos
 */
public class CuentasBancarias {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        //Crear clientel
        Cliente clientel = new Cliente(idCliente: 1, nombre: "Juan Carlos",
            direccion: "Calle Arrecife 23", telefono: "699699699");

        //Crear cliente2
        Cliente cliente2 = new Cliente();

        cliente2.setIdCliente(idCliente: 2);
        cliente2.setNombre(nombre: "Francisco");
        cliente2.setDireccion(direccion: "Calle La Liberta 13");
        cliente2.setTelefono(telefono: "688688688");

        //Crear CUENTA CORRIENTE para clientel
        CuentaCorriente cc = new CuentaCorriente(numeroDeCuenta: 1011, saldo: 4500,
            titular:clientel);

        //Crear CUENTA AHORRO para cliente2
        CuentaAhorro ca = new CuentaAhorro(numeroDeCuenta: 1022, saldo: 22000,
            titular:cliente2, interesVariable:0.02, saldoMinimo:20000);

        //Realizar operaciones cuentaCorriente con los metodos
        try {
            cc.ingresar(ingresarSaldo: 200); //Depositar
            cc.retirar(retirarSaldo: 1000); //retirar
            cc.actualizarSaldo(); //Actualizar Saldo con interes fijo
            System.out.println("Saldo final en la cuenta Corriente: " + cc.getSaldo());
        } catch (Exception e) {
            System.err.println("e: e.getMessage());
        }

        //Realizar operaciones cuentaAhorro con los metodos
        try {
            ca.ingresar(ingresarSaldo: 2500); //Depositar
            ca.retirar(retirarSaldo: 200); //retirar
            ca.actualizarSaldo(); //Actualizar Saldo con interes fijo
            System.out.println("Saldo final en la cuenta Ahorro: " + ca.getSaldo());
        } catch (Exception e) {
            System.err.println("e: e.getMessage());
        }
    }
}
```