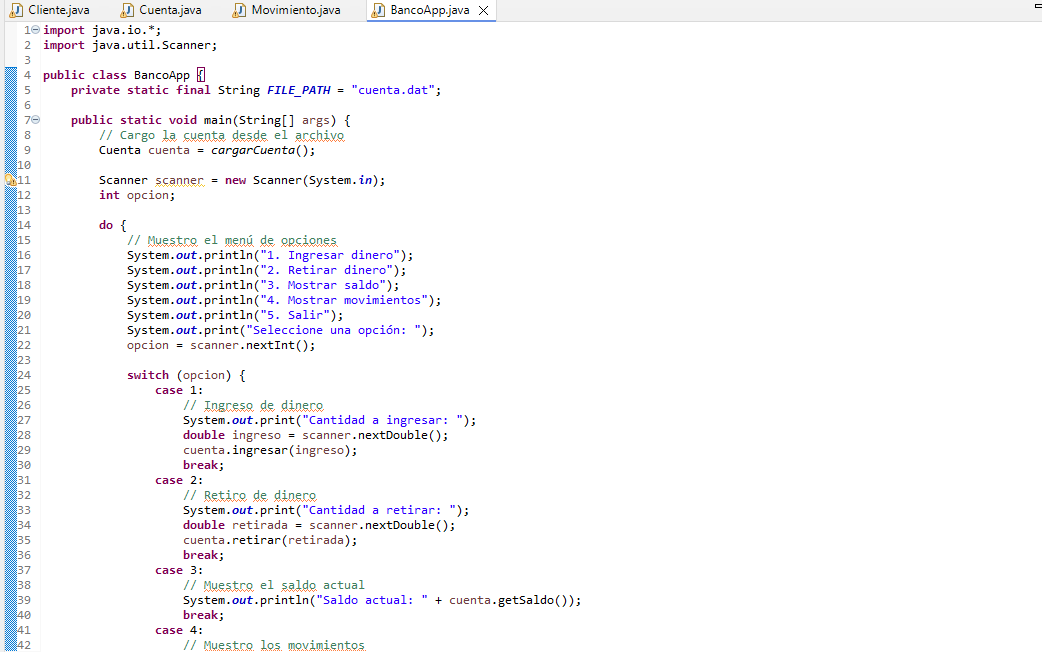
BancoApp:





El código de la clase BancoApp es una aplicación Java que gestiona una cuenta bancaria para un cliente. Aquí está la explicación de cada parte del código:

Importaciones: Se importan las clases necesarias para la entrada/salida de datos (java.io.\*) y para la entrada de datos desde el teclado (java.util.Scanner).

Constante FILE\_PATH: Se define una constante FILE\_PATH que contiene la ruta del archivo donde se guardará la cuenta (cuenta.dat).

Método main: Es el punto de entrada de la aplicación.

Se carga la cuenta desde el archivo llamando al método cargarCuenta.

Se crea un objeto Scanner para leer la entrada del usuario.

Se utiliza un bucle do-while para mostrar un menú de opciones al usuario y ejecutar la opción seleccionada.

Dependiendo de la opción seleccionada, se realizan diferentes operaciones sobre la cuenta (ingresar dinero, retirar dinero, mostrar saldo, mostrar movimientos).

Si el usuario selecciona la opción 5, se guarda la cuenta llamando al método guardarCuenta y se sale del bucle.

Método cargarCuenta: Carga la cuenta desde el archivo cuenta.dat.

Si el archivo existe, se lee el objeto Cuenta desde el archivo utilizando un ObjectInputStream.

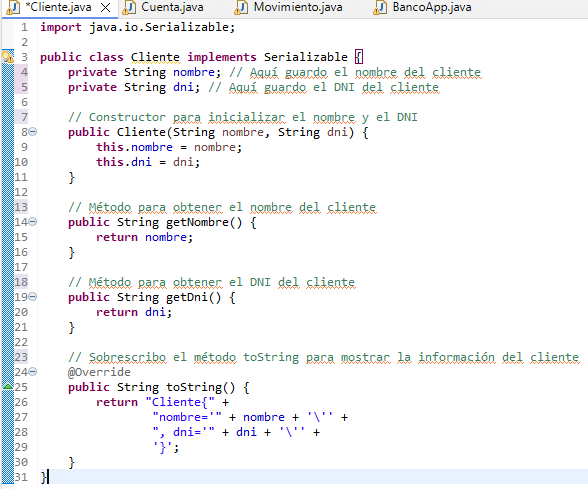
Si el archivo no existe o hay un error al leerlo, se crea una nueva cuenta con un cliente por defecto (Juan Perez).

Método guardarCuenta: Guarda la cuenta en el archivo cuenta.dat.

Se utiliza un ObjectOutputStream para escribir el objeto Cuenta en el archivo.

Si hay un error al escribir el archivo, se imprime la traza del error.

Cliente



La clase Cliente es una clase simple en Java que implementa la interfaz Serializable, lo que permite que sus instancias sean serializables. Aquí está la explicación de cada parte del código:

Importación: Importo java.io.Serializable, que es necesario para hacer que la clase sea serializable.

Declaración de la clase: Declaro la clase Cliente como public y especifico que implementa Serializable.

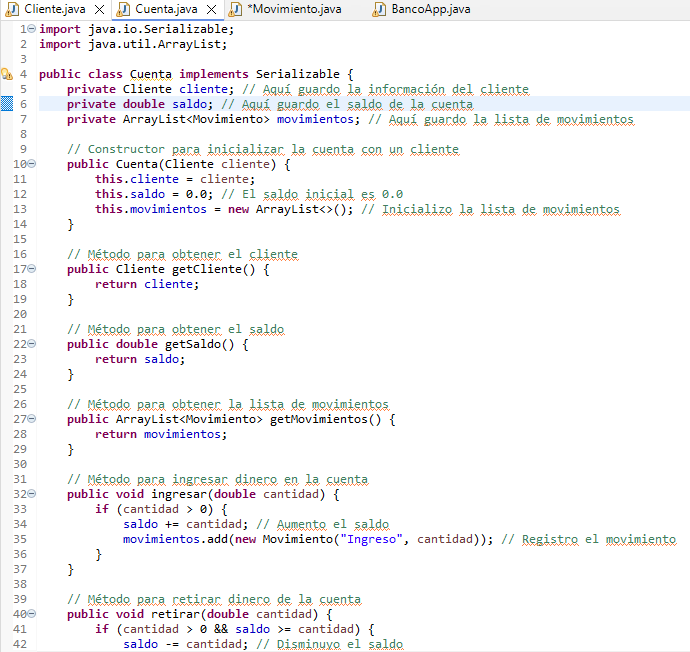
Atributos: La clase tiene dos atributos privados: nombre (nombre) y dni (número de identificación).

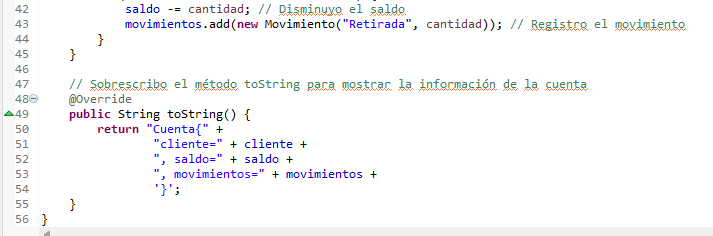
Constructor: El constructor inicializa los atributos nombre y dni con los valores proporcionados.

Getters: Proporciono métodos getter públicos para ambos atributos, nombre y dni.

Método toString: Sobrescribo el método toString para proporcionar una representación en cadena del objeto Cliente, que incluye los atributos nombre y dni.

Cuenta





La clase Cuenta es una clase en Java que implementa la interfaz Serializable, lo que permite que sus instancias sean serializables. Aquí está la explicación de cada parte del código:

Importaciones: Importo java.io.Serializable para la serialización y java.util.ArrayList para manejar una lista de movimientos.

Declaración de la clase: Declaro la clase Cuenta como public y especifico que implementa Serializable.

Atributos: La clase tiene tres atributos privados: cliente (de tipo Cliente), saldo (de tipo double) y movimientos (una lista de objetos Movimiento).

Constructor: El constructor inicializa el atributo cliente con el valor proporcionado, establece el saldo en 0.0 y crea una nueva lista vacía para movimientos.

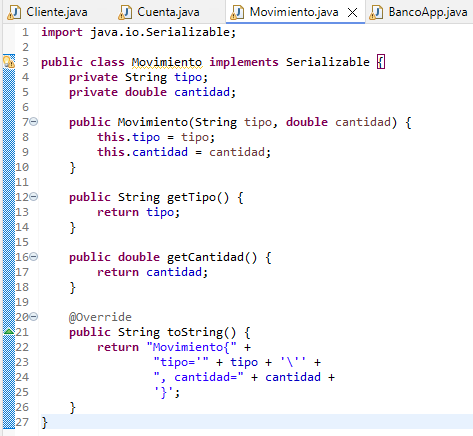
Getters: Proporciono métodos getter públicos para los atributos cliente, saldo y movimientos.

Método ingresar: Este método permite ingresar una cantidad de dinero en la cuenta. Si la cantidad es mayor que 0, se suma al saldo y se añade un nuevo movimiento de tipo "Ingreso" a la lista de movimientos.

Método retirar: Este método permite retirar una cantidad de dinero de la cuenta. Si la cantidad es mayor que 0 y el saldo es suficiente, se resta del saldo y se añade un nuevo movimiento de tipo "Retirada" a la lista de movimientos.

Método toString: Sobrescribo el método toString para proporcionar una representación en cadena del objeto Cuenta, que incluye los atributos cliente, saldo y movimientos.

Movimiento:



La clase Movimiento es una clase en Java que implementa la interfaz Serializable, lo que permite que sus instancias sean serializables. Aquí está la explicación de cada parte del código:

Importación: Importo java.io.Serializable, que es necesario para hacer que la clase sea serializable.

Declaración de la clase: Declaro la clase Movimiento como public y especifico que implementa Serializable.

Atributos: La clase tiene dos atributos privados: tipo (de tipo String) y cantidad (de tipo double).

Constructor: El constructor inicializa los atributos tipo y cantidad con los valores proporcionados.

Getters: Proporciono métodos getter públicos para los atributos tipo y cantidad.

Método toString: Sobrescribo el método toString para proporcionar una representación en cadena del objeto Movimiento, que incluye los atributos tipo y cantidad.

Reflexión:

En este programa que he desarrollado, la finalidad es gestionar las operaciones básicas de una cuenta bancaria: ingresar dinero, retirar fondos, consultar el saldo disponible y revisar el historial de movimientos. Además, he incorporado la capacidad de guardar y cargar el estado de la cuenta en un archivo, lo que permite que los datos de la cuenta persistan entre sesiones. A continuación, te voy a explicar cada sección del programa en detalle, en primera persona, reflexionando sobre las decisiones que tomé al escribir cada parte.

**1. Clase BancoApp y la Estructura Principal**

En la clase BancoApp, defino el flujo de ejecución principal del programa, que es el núcleo de interacción con el usuario. Aquí utilizo un menú de opciones que permite realizar diferentes acciones sobre la cuenta bancaria.

Al inicio, tengo una constante FILE\_PATH que define la ruta del archivo donde voy a guardar la cuenta. Esto lo hice porque es importante tener un lugar fijo donde guardar los datos, y me pareció práctico definir esta constante para evitar errores al repetir el nombre del archivo en varias partes del código.

**Método main**

El método main es el punto de entrada del programa. Lo primero que hago es llamar al método cargarCuenta() para obtener el estado de la cuenta si ya existía en el archivo. Si no existía, simplemente se crea una nueva cuenta para empezar.

Decidí que el uso de un bucle do-while sería adecuado para permitir al usuario interactuar repetidamente con el sistema hasta que decida salir. Aquí uso la clase Scanner para capturar la entrada del usuario, ya que es sencilla y eficaz para este tipo de programas de consola.

**2. Menú de Opciones**

Implemento un menú que ofrece cinco opciones al usuario:

* **Ingresar dinero**: Cuando el usuario selecciona esta opción, le pido que ingrese una cantidad, y llamo al método ingresar() de la clase Cuenta para que el saldo de la cuenta aumente. Me parece importante validar que solo se ingresen cantidades positivas.
* **Retirar dinero**: Para retirar dinero, también solicito una cantidad y luego verifico que la cantidad a retirar sea positiva y que no exceda el saldo disponible. Si se cumplen estas condiciones, uso el método retirar() de la cuenta para realizar la operación. De nuevo, decidí incluir validaciones básicas para evitar errores.
* **Mostrar saldo**: Aquí, simplemente accedo al saldo de la cuenta mediante el método getSaldo(). Este método es muy simple, pero su inclusión en el menú es esencial para que el usuario pueda consultar su balance actual en cualquier momento.
* **Mostrar movimientos**: En este caso, llamo al método getMovimientos() para obtener una lista de las transacciones realizadas. Decidí que los movimientos fueran una lista de objetos de tipo Movimiento, ya que me permite almacenar tanto el tipo de transacción (ingreso o retirada) como la cantidad involucrada.
* **Salir**: Si el usuario selecciona esta opción, llamo al método guardarCuenta() para asegurar que el estado de la cuenta se guarde en el archivo. De esta manera, cuando se ejecute el programa nuevamente, la cuenta estará en el mismo estado que cuando el usuario la dejó.

**3. Métodos cargarCuenta() y guardarCuenta()**

Decidí manejar el almacenamiento de la cuenta en un archivo binario utilizando la serialización. Esto lo hago porque me permite almacenar no solo los datos del saldo, sino también todos los movimientos y la información del cliente de manera sencilla.

**cargarCuenta()**

Este método verifica si existe un archivo llamado "cuenta.dat". Si existe, lo lee utilizando un ObjectInputStream, y trato de deserializar la cuenta desde ese archivo. Si ocurre algún error, capturo las excepciones pertinentes y creo una nueva cuenta desde cero.

El propósito de hacer esto es asegurar que el usuario nunca se quede sin una cuenta, incluso si ocurre algún fallo en la lectura del archivo. Esta estrategia me da flexibilidad para manejar posibles problemas, como un archivo corrupto o no encontrado, sin interrumpir la experiencia del usuario.

**guardarCuenta()**

Por otro lado, al terminar el programa, uso el método guardarCuenta() para serializar el objeto Cuenta y almacenarlo en el archivo "cuenta.dat". La razón por la que uso un ObjectOutputStream aquí es que facilita la escritura del objeto completo en un solo paso, guardando tanto el saldo como los movimientos y la información del cliente.

**4. Clase Cliente**

El objeto Cliente representa al titular de la cuenta. Tiene dos propiedades esenciales: el nombre del cliente y su DNI (Documento Nacional de Identidad).

Cuando diseñé esta clase, la hice Serializable para poder almacenarla junto con la cuenta. Opté por un diseño simple aquí, con un constructor que inicializa los atributos del cliente, y métodos getNombre() y getDni() que devuelven esta información. Aunque es una clase sencilla, es fundamental para tener una estructura clara en torno al concepto de titular de la cuenta, que es algo central en cualquier sistema bancario.

**5. Clase Cuenta**

Aquí es donde se gestionan las operaciones principales relacionadas con el dinero. La clase Cuenta contiene la información sobre el saldo y los movimientos realizados, además de un objeto Cliente que asocia la cuenta con una persona específica.

**Atributos de Cuenta**

* **Cliente**: Cada cuenta tiene asociado un cliente, que es el titular de la misma. Esta relación es importante para que siempre se pueda identificar a quién pertenece la cuenta.
* **Saldo**: Almaceno el saldo actual de la cuenta en un atributo de tipo double. Me aseguré de que las operaciones que modifiquen este saldo (como ingresos y retiradas) estén bien controladas.
* **Movimientos**: Almaceno todos los movimientos realizados en una lista de objetos Movimiento. Este enfoque me permite guardar tanto los ingresos como las retiradas de manera ordenada y detallada.

**Métodos de Cuenta**

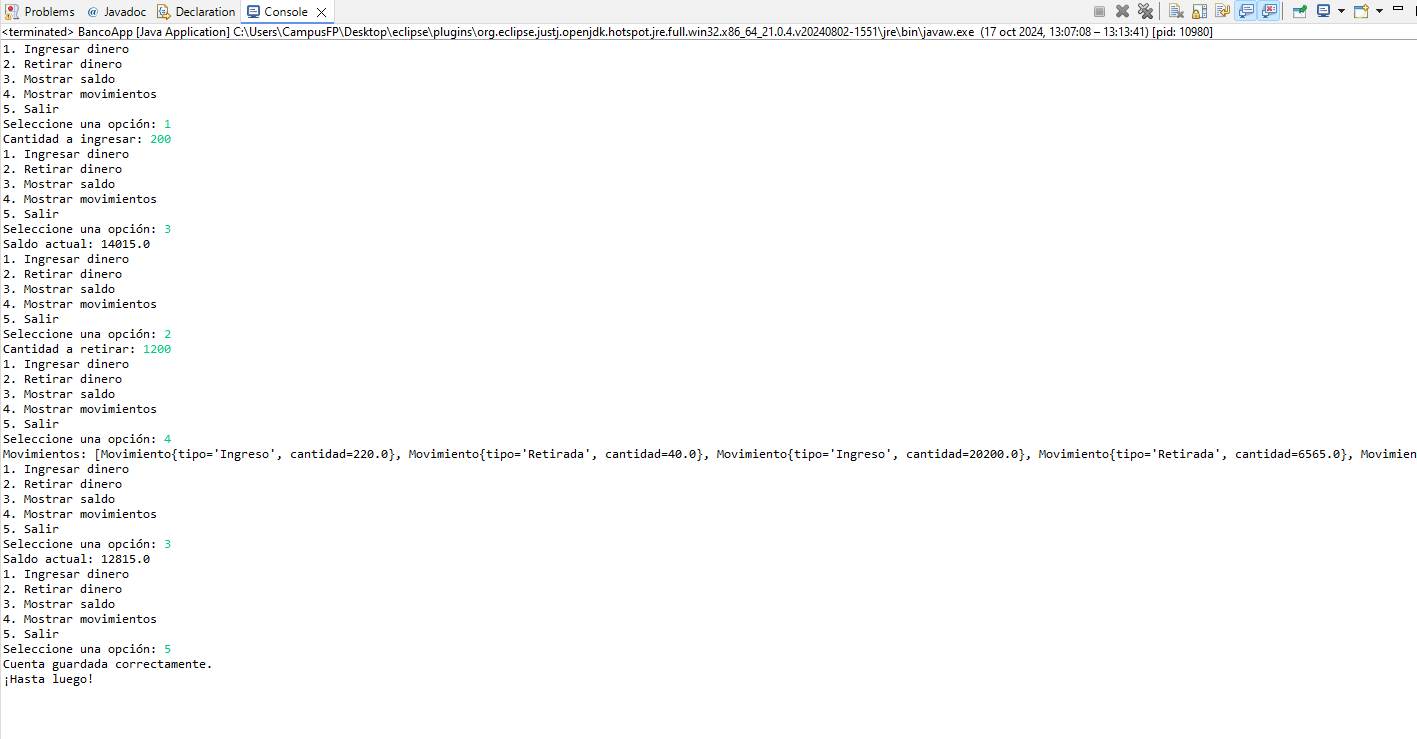
* **ingresar()**: Este método recibe una cantidad y la añade al saldo si es positiva. Cada vez que se hace un ingreso, también añado un nuevo objeto Movimiento a la lista, con el tipo de operación y la cantidad. Esto me asegura que todo quede registrado.
* **retirar()**: Similar al método de ingreso, pero aquí verifico que la cantidad a retirar sea positiva y que el saldo sea suficiente para cubrir la retirada. Si se cumple, se realiza la operación y se registra en los movimientos.

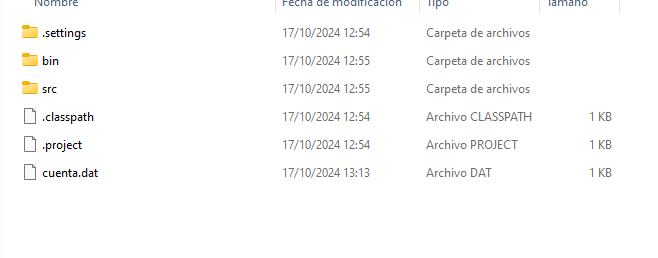
**6. Clase Movimiento**

Cada vez que se realiza una operación en la cuenta, creo un nuevo objeto Movimiento que registra el tipo de operación ("Ingreso" o "Retirada") y la cantidad correspondiente. Hice esta clase Serializable para poder guardarla junto con la cuenta. De esta forma, siempre puedo llevar un control detallado de cada operación realizada en la cuenta.

En resumen, con este programa, he tratado de crear una simulación básica de una cuenta bancaria que permita realizar las operaciones fundamentales de una manera intuitiva. Al usar clases como Cliente, Cuenta y Movimiento, encapsulo adecuadamente la lógica de la aplicación. Además, con la serialización, aseguro que los datos persistan entre las sesiones, lo que mejora la experiencia del usuario al no perder su información.

Prueba de funcionamiento y ejecución



Se crea Cuenta.dat correctamente  


Anexo  
BancoApp

**import** java.io.\*;

**import** java.util.Scanner;

**public** **class** BancoApp {

**private** **static** **final** String ***FILE\_PATH*** = "cuenta.dat";

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

Cuenta cuenta = *cargarCuenta*();

Scanner scanner = **new** Scanner(System.***in***);

**int** opcion;

**do** {

System.***out***.println("1. Ingresar dinero");

System.***out***.println("2. Retirar dinero");

System.***out***.println("3. Mostrar saldo");

System.***out***.println("4. Mostrar movimientos");

System.***out***.println("5. Salir");

System.***out***.print("Seleccione una opción: ");

opcion = scanner.nextInt();

**switch** (opcion) {

**case** 1:

System.***out***.print("Cantidad a ingresar: ");

**double** ingreso = scanner.nextDouble();

cuenta.ingresar(ingreso);

**break**;

**case** 2:

System.***out***.print("Cantidad a retirar: ");

**double** retirada = scanner.nextDouble();

cuenta.retirar(retirada);

**break**;

**case** 3:

System.***out***.println("Saldo actual: " + cuenta.getSaldo());

**break**;

**case** 4:

System.***out***.println("Movimientos: " + cuenta.getMovimientos());

**break**;

**case** 5:

*guardarCuenta*(cuenta);

System.***out***.println("¡Hasta luego!");

**break**;

**default**:

System.***out***.println("Opción no válida.");

}

} **while** (opcion != 5);

}

**private** **static** Cuenta cargarCuenta() {

File file = **new** File(***FILE\_PATH***);

**if** (file.exists()) {

**try** (ObjectInputStream ois = **new** ObjectInputStream(**new** FileInputStream(file))) {

**return** (Cuenta) ois.readObject();

} **catch** (IOException | ClassNotFoundException e) {

e.printStackTrace();

}

}

**return** **new** Cuenta(**new** Cliente("Juan Perez", "12345678A"));

}

**private** **static** **void** guardarCuenta(Cuenta cuenta) {

**try** (ObjectOutputStream oos = **new** ObjectOutputStream(**new** FileOutputStream(***FILE\_PATH***))) {

oos.writeObject(cuenta);

System.***out***.println("Cuenta guardada correctamente.");

} **catch** (IOException e) {

e.printStackTrace();

}

}

}

Cliente

**import** java.io.Serializable;

**public** **class** Cliente **implements** Serializable {

**private** String nombre;

**private** String dni;

**public** Cliente(String nombre, String dni) {

**this**.nombre = nombre;

**this**.dni = dni;

}

**public** String getNombre() {

**return** nombre;

}

**public** String getDni() {

**return** dni;

}

@Override

**public** String toString() {

**return** "Cliente{" +

"nombre='" + nombre + '\'' +

", dni='" + dni + '\'' +

'}';

}

}

Cuenta:

**import** java.io.Serializable;

**import** java.util.ArrayList;

**public** **class** Cuenta **implements** Serializable {

**private** Cliente cliente;

**private** **double** saldo;

**private** ArrayList<Movimiento> movimientos;

**public** Cuenta(Cliente cliente) {

**this**.cliente = cliente;

**this**.saldo = 0.0;

**this**.movimientos = **new** ArrayList<>();

}

**public** Cliente getCliente() {

**return** cliente;

}

**public** **double** getSaldo() {

**return** saldo;

}

**public** ArrayList<Movimiento> getMovimientos() {

**return** movimientos;

}

**public** **void** ingresar(**double** cantidad) {

**if** (cantidad > 0) {

saldo += cantidad;

movimientos.add(**new** Movimiento("Ingreso", cantidad));

}

}

**public** **void** retirar(**double** cantidad) {

**if** (cantidad > 0 && saldo >= cantidad) {

saldo -= cantidad;

movimientos.add(**new** Movimiento("Retirada", cantidad));

}

}

@Override

**public** String toString() {

**return** "Cuenta{" +

"cliente=" + cliente +

", saldo=" + saldo +

", movimientos=" + movimientos +

'}';

}

}

Movimiento:

**import** java.io.Serializable;

**public** **class** Movimiento **implements** Serializable {

**private** String tipo;

**private** **double** cantidad;

**public** Movimiento(String tipo, **double** cantidad) {

**this**.tipo = tipo;

**this**.cantidad = cantidad;

}

**public** String getTipo() {

**return** tipo;

}

**public** **double** getCantidad() {

**return** cantidad;

}

@Override

**public** String toString() {

**return** "Movimiento{" +

"tipo='" + tipo + '\'' +

", cantidad=" + cantidad +

'}';

}

}