ESCUELA POLITÉCNICA NACIONAL

ESCUELA DE FORMACIÓN DE TECNÓLOGOS

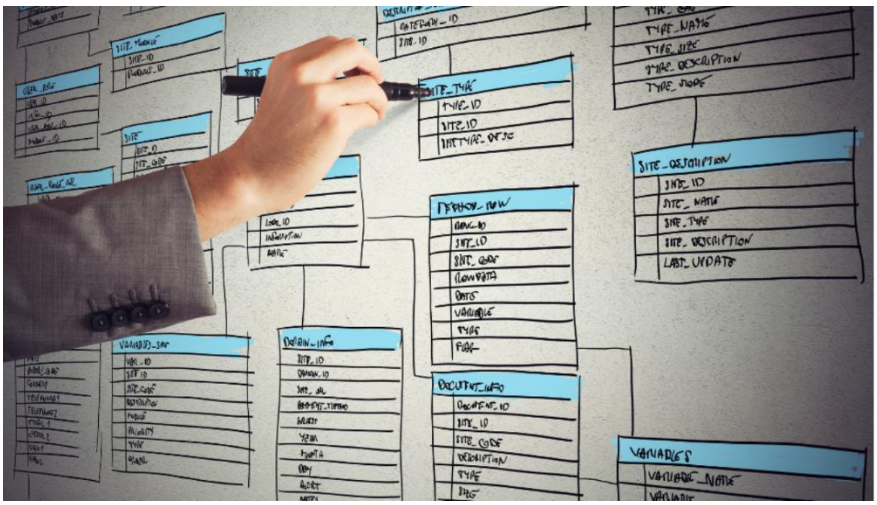
PROGRAMACIÓN ORIENTADA A OBJETOS

PROFESOR:	Ing. Yadira Franco R
PERÍODO ACADÉMICO:	2025-A

TAREA 6 – Grupal 2 integrantes

TÍTULO:

INVESTIGACIÓN Y PRACTICA



Estudiante

Juan Lucero, Brandon Huera

2025-A

Sistema Bancario: Uso de Clases Abstractas y Jerarquías

Link al repositorio de github:

https://github.com/JuanL525/CuentaBancariaPOO.git

1. Clase Abstracta: `Persona`

Definición: Una clase abstracta no puede instanciarse directamente. Sirve como modelo base para otras clases y puede contener métodos abstractos (sin cuerpo) que deben ser implementados por sus subclases.

Atributos comunes sugeridos:

- protected String nombre;

- protected String cedula;

- protected String direccion;

- protected String telefono;

Métodos comunes:

public void actualizarDatos(String direccion, String telefono) { ... }

public abstract void mostrarRol();

Investigación sugerida:

- ¿Qué es una clase abstracta en Java?

Una **clase abstracta** en Java es una clase que **no se puede instanciar directamente** y está diseñada para ser **extendida por otras clases**. Sirve como una plantilla

- ¿Por qué no se puede instanciar una clase abstracta?

No se puede instanciar una clase abstracta porque **no está completamente definida**. Puede tener métodos sin implementación, y como tal, no tendría sentido crear objetos de una clase incompleta. El objetivo es que una **subclase proporcione las implementaciones** necesarias para los métodos abstractos antes de que pueda usarse.

- ¿Debe el constructor de una clase abstracta ser también abstracto?

Una clase abstracta **puede tener un constructor**, pero **no puede ser abstracto**. Su propósito es inicializar los atributos comunes a todas las subclases. El constructor se llama **implícitamente** cuando una subclase concreta crea un objeto.

- ¿Cuándo usar y cuándo no usar clases abstractas?

Cuando usar clases abstractas:

- Cuando varias clases comparten comportamiento **común** pero también tienen comportamientos específicos.
- Cuando deseas **forzar a las subclases a implementar ciertos métodos** (usando métodos abstractos).
- Cuando quieres **proveer una base común con métodos reutilizables** y atributos compartidos.
- En el diseño de **jerarquías de clases** (como `Animal`, `Vehículo`, `Empleado`, etc.).

Cuando no usar clases abstractas:

- Cuando **no hay una relación clara de herencia** entre las clases.
- Si necesitas heredar de múltiples tipos: Java **no permite herencia múltiple con clases**, pero **sí con interfaces**.
- Cuando todas las implementaciones serán completas: en ese caso, **usa una clase concreta**.
- Cuando el código debe ser altamente **flexible o desacoplado**: puede ser mejor usar **interfaces**.

2. Modificadores de acceso: `private` vs. `protected`

- private: Solo accesible dentro de la misma clase.

- protected: Accesible dentro de la misma clase, subclases, y paquete.

- public: Accesible desde cualquier lugar.

Uso sugerido:

- Usa protected en clases abstractas para que los atributos sean accesibles por las subclases.

- Usa private cuando quieras ocultar atributos que solo se modifiquen mediante getters/setters.

3. Subclase: `Cliente extends Persona`

Responsabilidad: Consumidor de productos bancarios.

Métodos específicos:

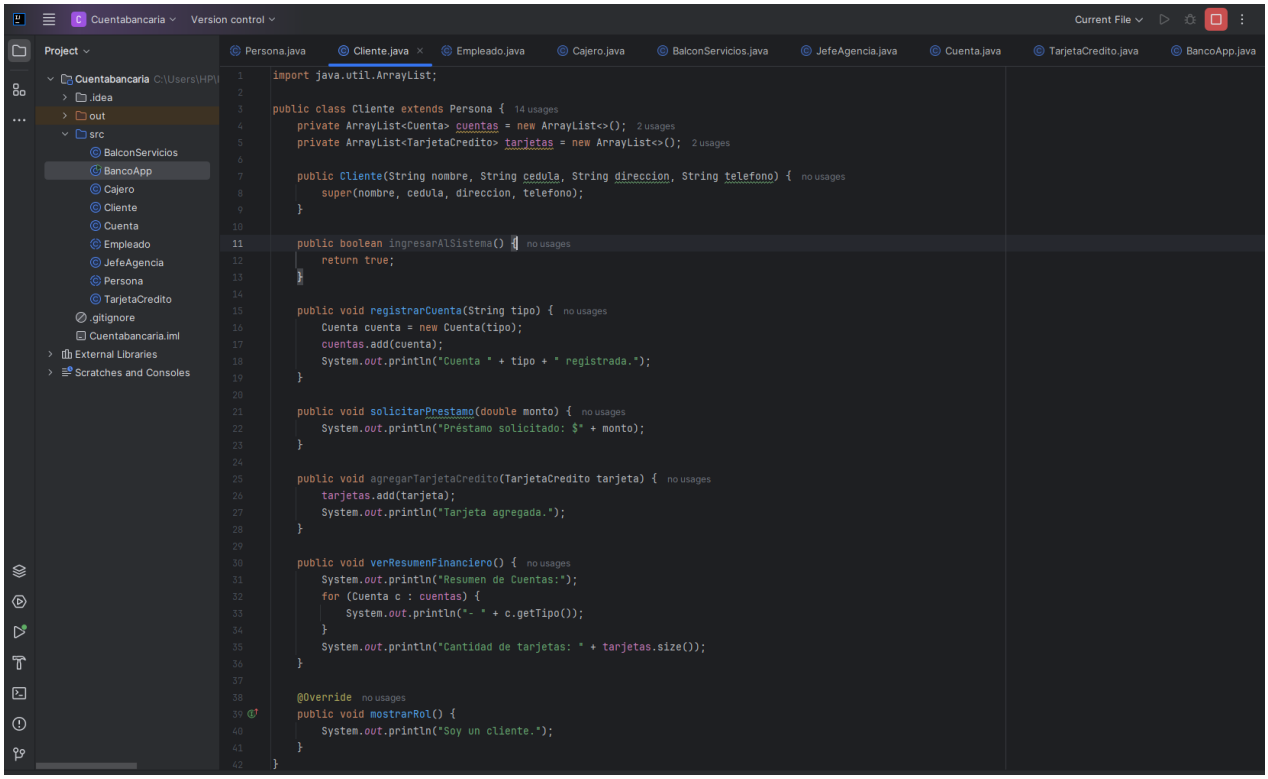
+ ingresarAlSistema(): boolean

+ registrarCuenta(tipo: String): void

+ solicitarPrestamo(monto: double): void

+ agregarTarjetaCredito(tarjeta: TarjetaCredito): void

+ verResumenFinanciero(): void



Uso: Se pueden crear objetos Cliente en el main() utilizando Scanner para ingresar datos como nombre, cédula y tipo de cuenta.

4. Subclase: `Empleado extends Persona`

Responsabilidad: Gestionar productos y datos bancarios de clientes.

- Cajero

- BalconServicios

- JefeAgencia (Gerente)

Métodos sugeridos para Empleado:

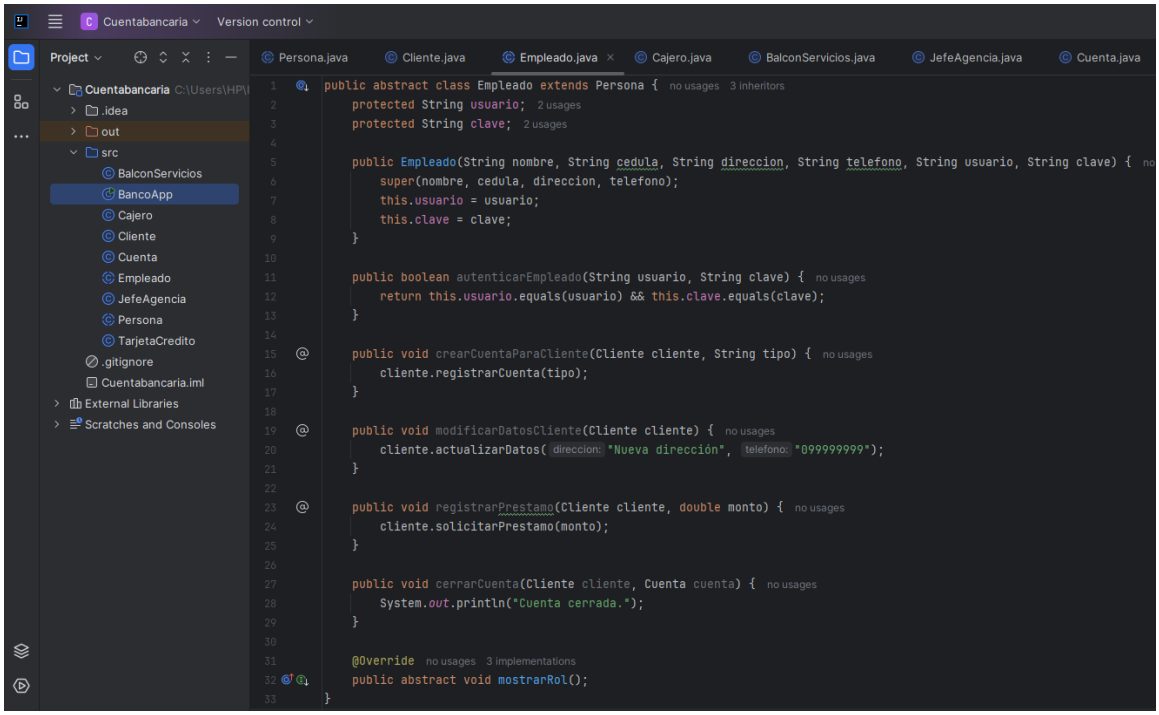
+ autenticarEmpleado(usuario: String, clave: String): boolean

+ crearCuentaParaCliente(cliente: Cliente, tipo: String): void

+ modificarDatosCliente(cliente: Cliente): void

+ registrarPrestamo(cliente: Cliente, monto: double): void

+ cerrarCuenta(cliente: Cliente, cuenta: Cuenta): void



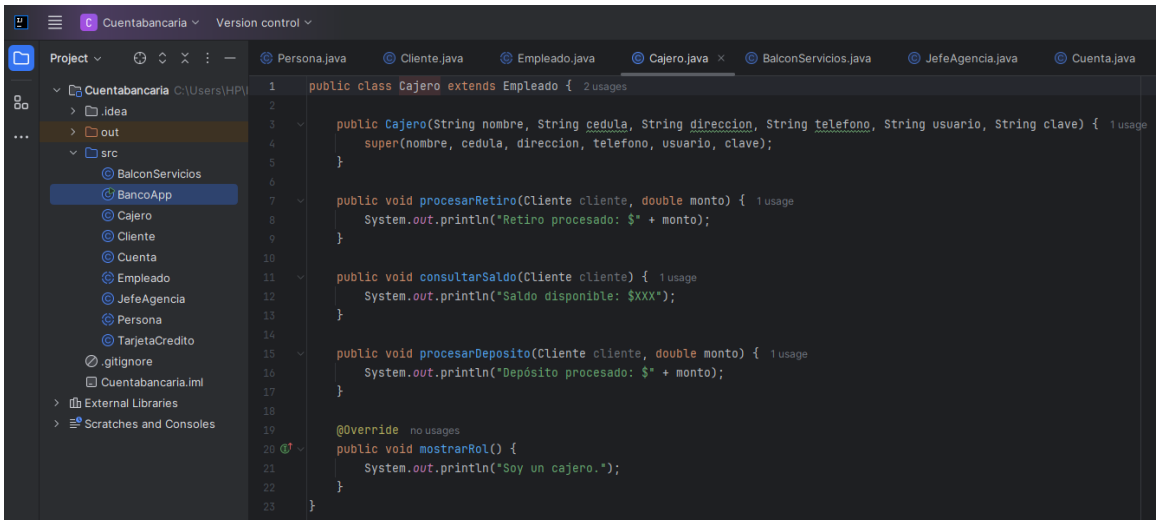
Métodos adicionales sugeridos por rol:

Para Cajero:

+ procesarRetiro(cliente: Cliente, monto: double): void

+ consultarSaldo(cliente: Cliente): void

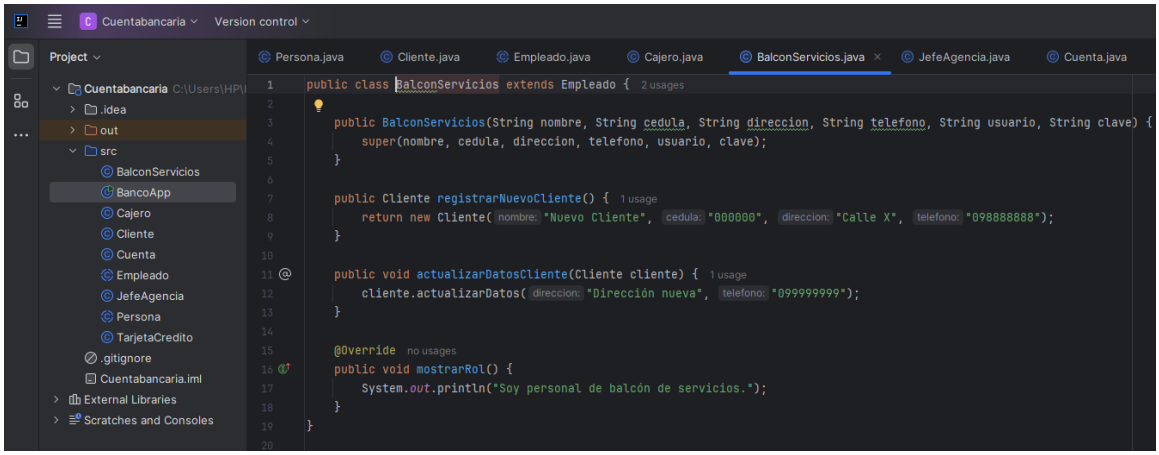
+ procesarDeposito(cliente: Cliente, monto: double): void



Para BalconServicios:

+ registrarNuevoCliente(): Cliente

+ actualizarDatosCliente(cliente: Cliente): void

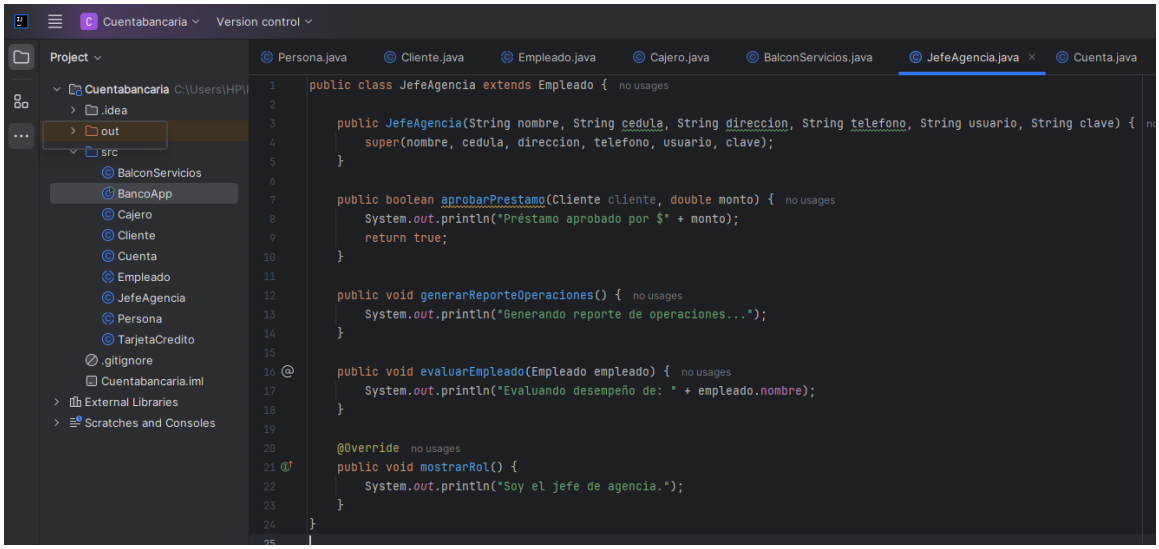


Para JefeAgencia:

+ aprobarPrestamo(cliente: Cliente, monto: double): boolean

+ generarReporteOperaciones(): void

+ evaluarEmpleado(empleado: Empleado): void



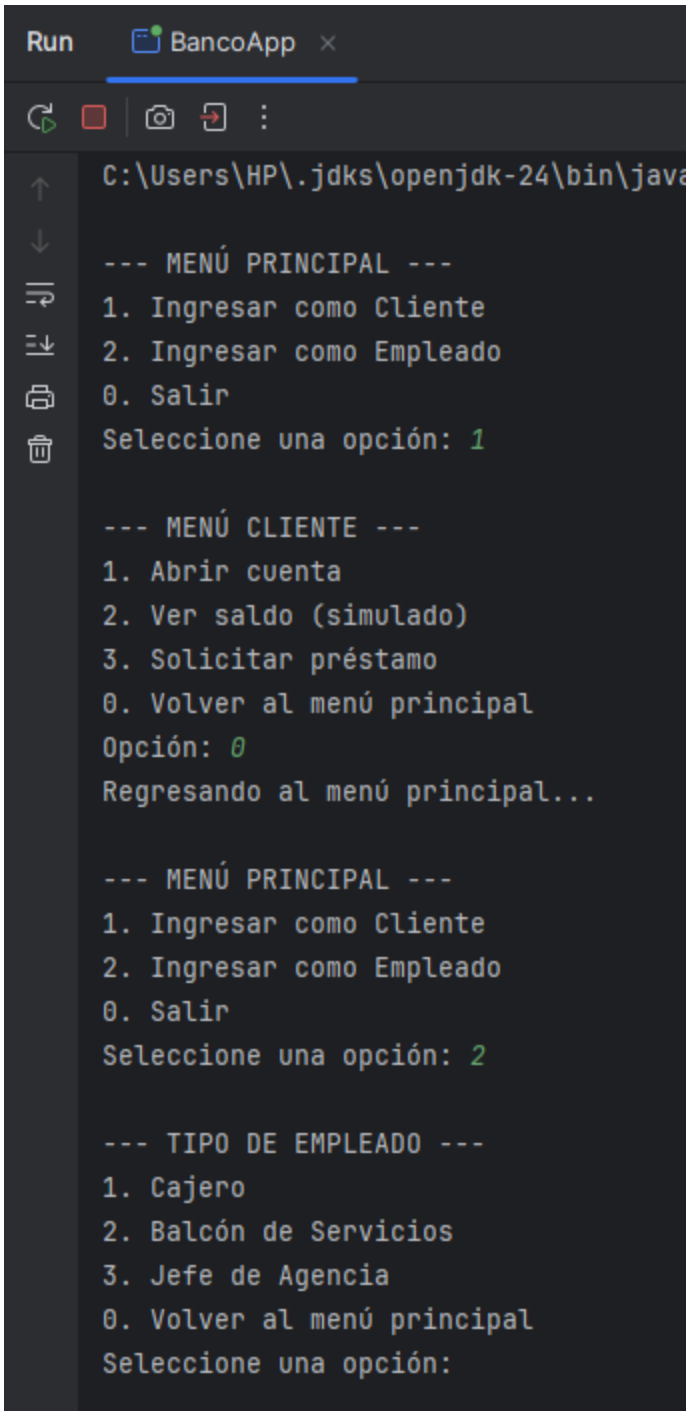
5. Flujo sugerido para una implementación con menú y objetos

1. Mostrar menú principal:

   - Registrar cliente

   - Ingresar como cliente

   - Ingresar como empleado (cajero, balcón, jefe agencia)

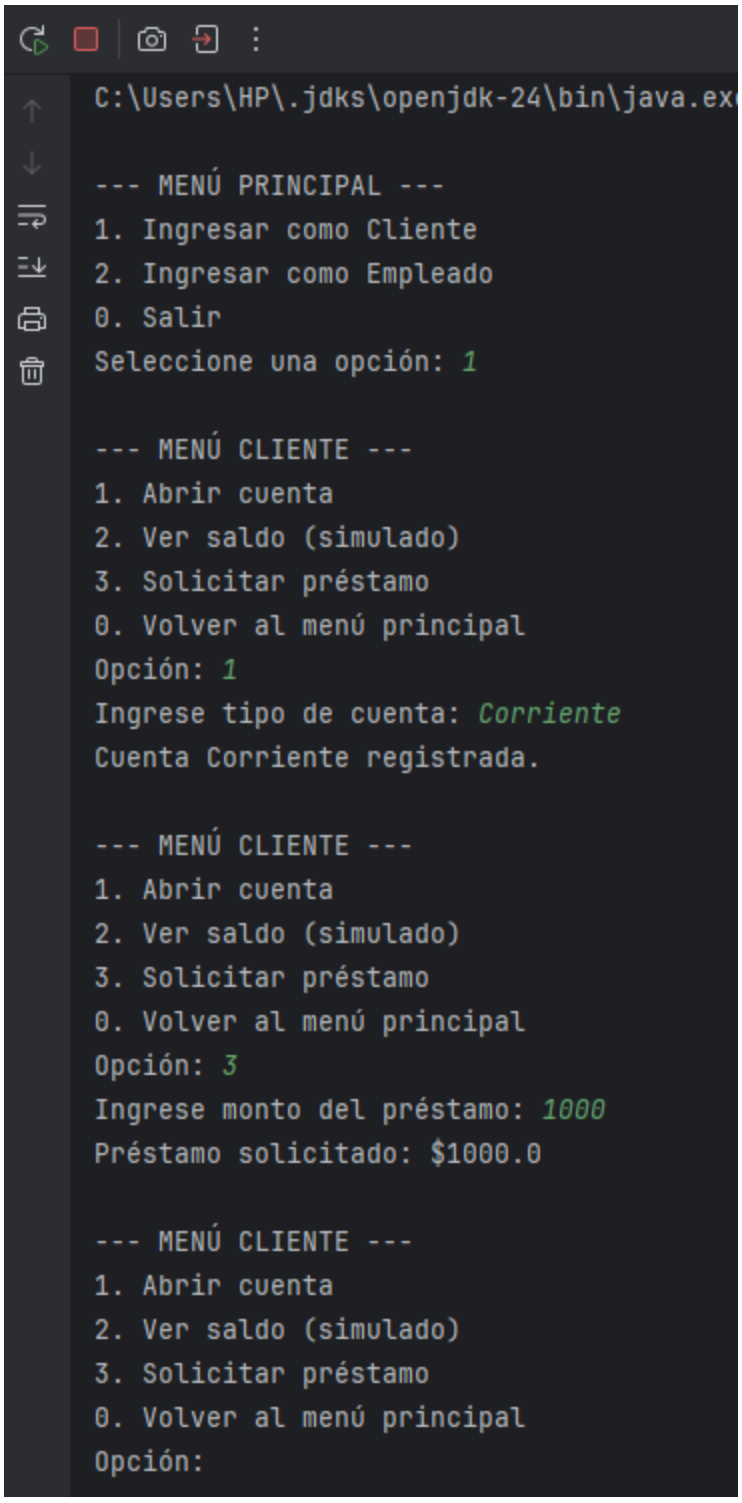


2. Usar Scanner para capturar datos.

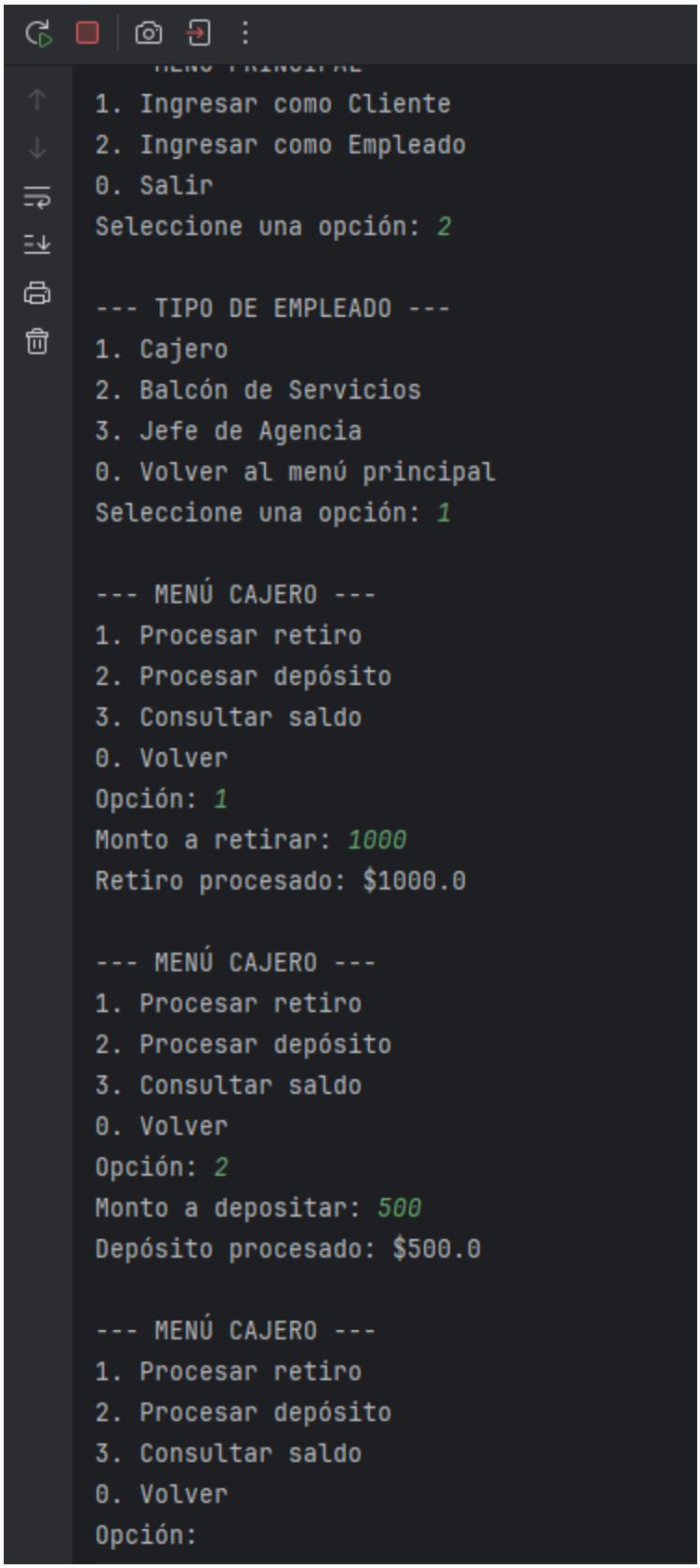
3. Crear objetos de tipo Cliente o Empleado según selección.

4. Permitir navegación entre funciones según el rol:

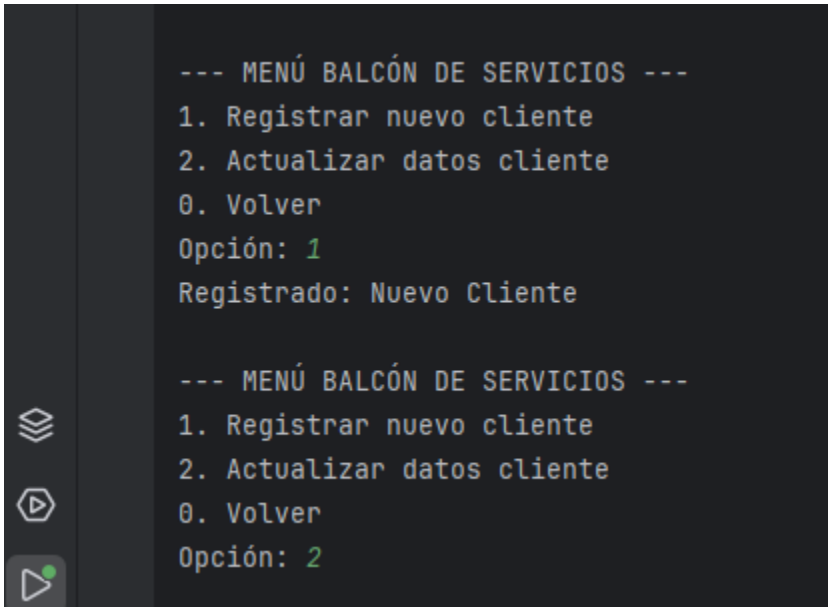
- Cliente: abrir cuenta, ver saldo, solicitar préstamo



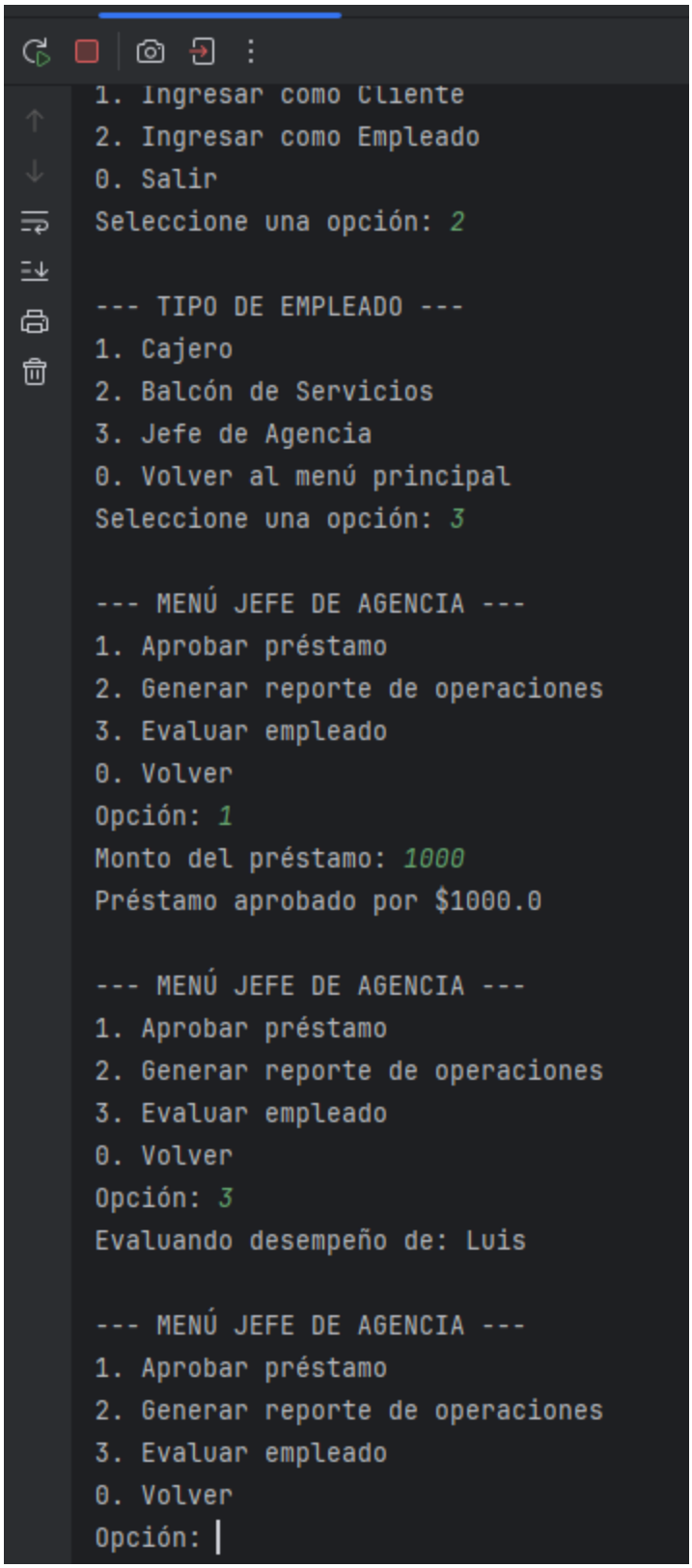
- Cajero: retiro, depósito


1. Ingresar como Cliente
2. Ingresar como Empleado
0. Salir
Seleccione una opción: 2
--- TIPO DE EMPLEADO ---
1. Cajero
2. Balcón de Servicios
3. Jefe de Agencia
0. Volver al menú principal
Seleccione una opción: 1
--- MENÚ CAJERO ---
1. Procesar retiro
2. Procesar depósito
3. Consultar saldo
0. Volver
Opción: 1
Monto a retirar: 1000
Retiro procesado: $1000.0
--- MENÚ CAJERO ---
1. Procesar retiro
2. Procesar depósito
3. Consultar saldo
0. Volver
Opción: 2
Monto a depositar: 500
Depósito procesado: $500.0
--- MENÚ CAJERO ---
1. Procesar retiro
2. Procesar depósito
3. Consultar saldo
0. Volver
Opción:

- Balcón: registro, modificación de datos



- Jefe de Agencia: aprobación y reportes



5. Aplicar encapsulamiento (private, protected) correctamente para proteger los atributos.

La implementación con Scanner y menús mejora la interacción con el usuario y refuerza el entendimiento práctico del tema.

Subir al Git hub – código e informe -