

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 subclasses.

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 subclasses. 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:

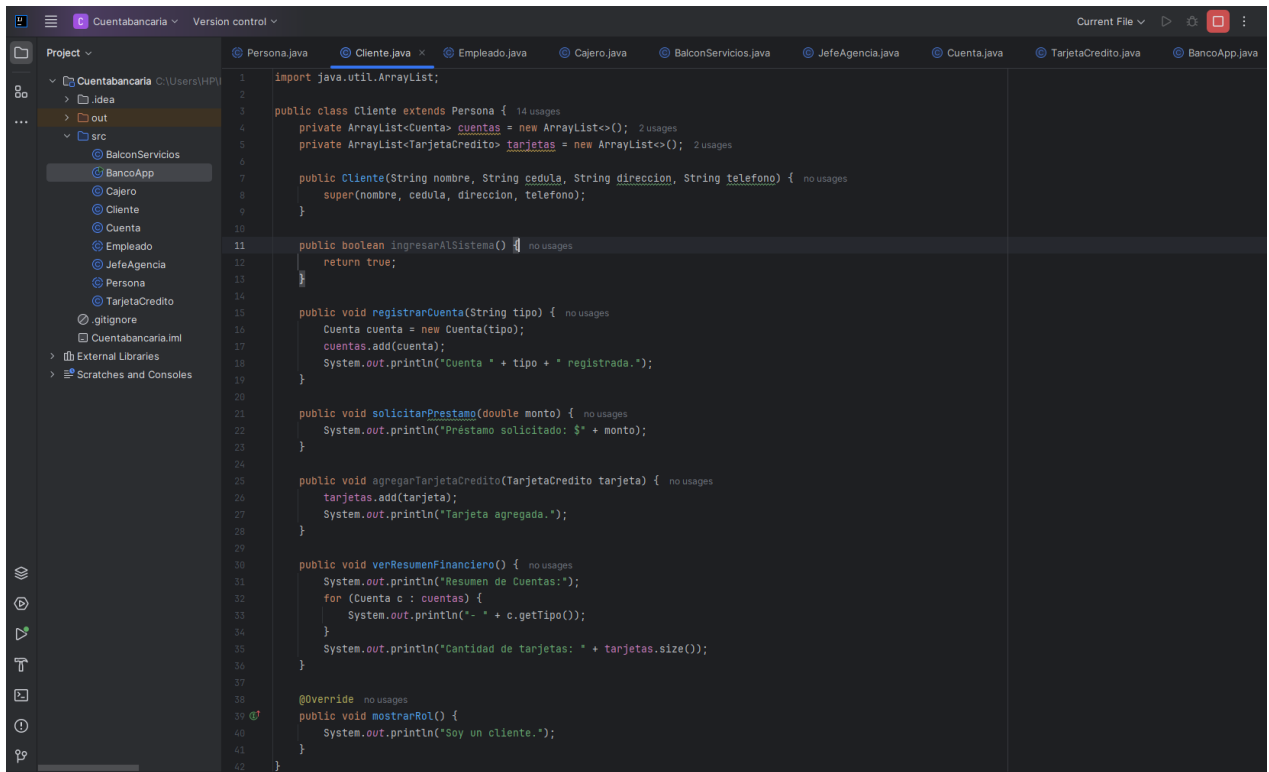
- Usa `protected` en clases abstractas para que los atributos sean accesibles por las subclases.
- Usa `private` cuando quieras ocultar atributos que solo se modifiquen mediante getters/setters.

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

```

1 public abstract class Empleado extends Persona {
2     protected String usuario;
3     protected String clave;
4
5     public Empleado(String nombre, String cedula, String direccion, String telefono, String usuario, String clave) {
6         super(nombre, cedula, direccion, telefono);
7         this.usuario = usuario;
8         this.clave = clave;
9     }
10
11     public boolean autenticarEmpleado(String usuario, String clave) {
12         return this.usuario.equals(usuario) && this.clave.equals(clave);
13     }
14
15     public void crearCuentaParaCliente(Cliente cliente, String tipo) {
16         cliente.registrarCuenta(tipo);
17     }
18
19     public void modificarDatosCliente(Cliente cliente) {
20         cliente.actualizarDatos("Nueva dirección", "099999999");
21     }
22
23     public void registrarPrestamo(Cliente cliente, double monto) {
24         cliente.solicitarPrestamo(monto);
25     }
26
27     public void cerrarCuenta(Cliente cliente, Cuenta cuenta) {
28         System.out.println("Cuenta cerrada.");
29     }
30
31     @Override
32     public abstract void mostrarRol();
33 }

```

Métodos adicionales sugeridos por rol:

Para Cajero:

- + procesarRetiro(cliente: Cliente, monto: double): void
- + consultarSaldo(cliente: Cliente): void
- + procesarDeposito(cliente: Cliente, monto: double): void

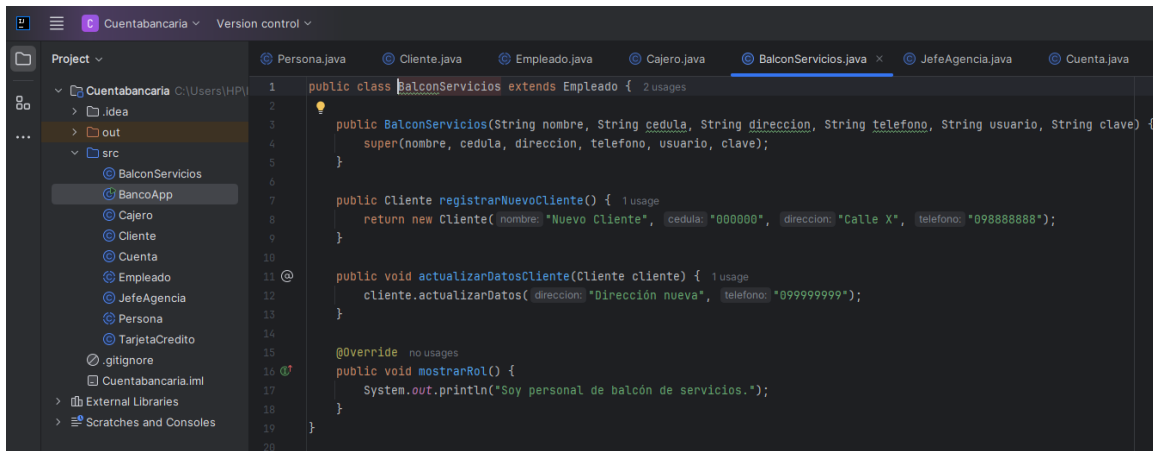
```

1 public class Cajero extends Empleado {
2     public Cajero(String nombre, String cedula, String direccion, String telefono, String usuario, String clave) {
3         super(nombre, cedula, direccion, telefono, usuario, clave);
4     }
5
6     public void procesarRetiro(Cliente cliente, double monto) {
7         System.out.println("Retiro procesado: $" + monto);
8     }
9
10    public void consultarSaldo(Cliente cliente) {
11        System.out.println("Saldo disponible: $XXX");
12    }
13
14    public void procesarDeposito(Cliente cliente, double monto) {
15        System.out.println("Depósito procesado: $" + monto);
16    }
17
18    @Override
19    public void mostrarRol() {
20        System.out.println("Soy un cajero.");
21    }
22 }

```

Para BalconServicios:

- + registrarNuevoCliente(): Cliente
- + actualizarDatosCliente(cliente: Cliente): void



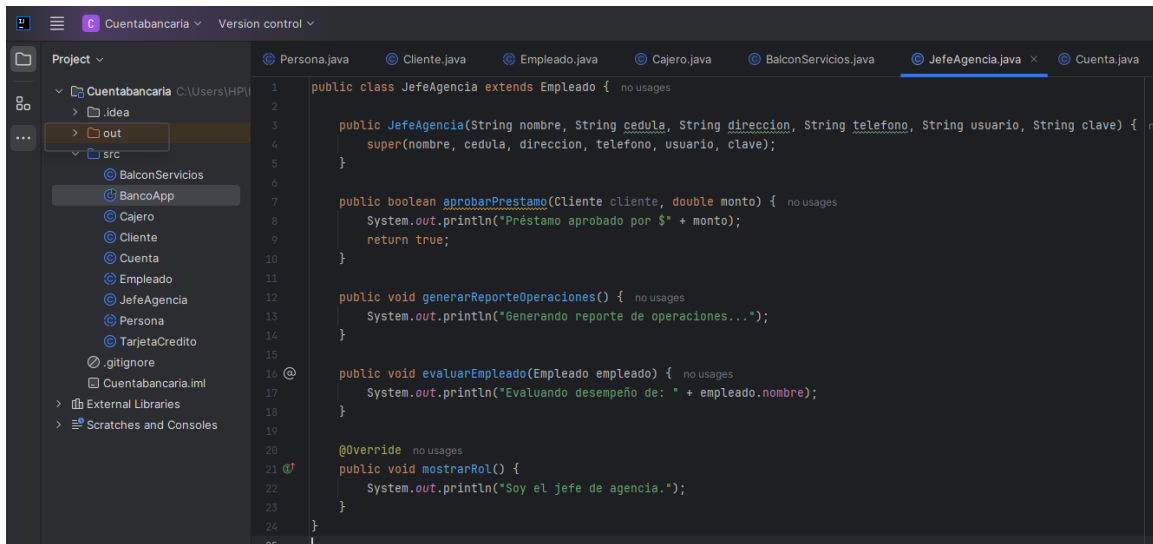
```
1 public class BalconServicios extends Empleado { 2 usages
2
3     public BalconServicios(String nombre, String cedula, String direccion, String telefono, String usuario, String clave) {
4         super(nombre, cedula, direccion, telefono, usuario, clave);
5     }
6
7     public Cliente registrarNuevoCliente() { 1 usage
8         return new Cliente(nombre: "Nuevo Cliente", cedula: "000000", direccion: "Calle X", telefono: "098888888");
9     }
10
11     public void actualizarDatosCliente(Cliente cliente) { 1 usage
12         cliente.actualizarDatos(direccion: "Dirección nueva", telefono: "099999999");
13     }
14
15     @Override no usages
16     public void mostrarRol() {
17         System.out.println("Soy personal de balcón de servicios.");
18     }
19 }
```

Para JefeAgencia:

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

+ generarReporteOperaciones(): void

+ evaluarEmpleado(empleado: Empleado): void

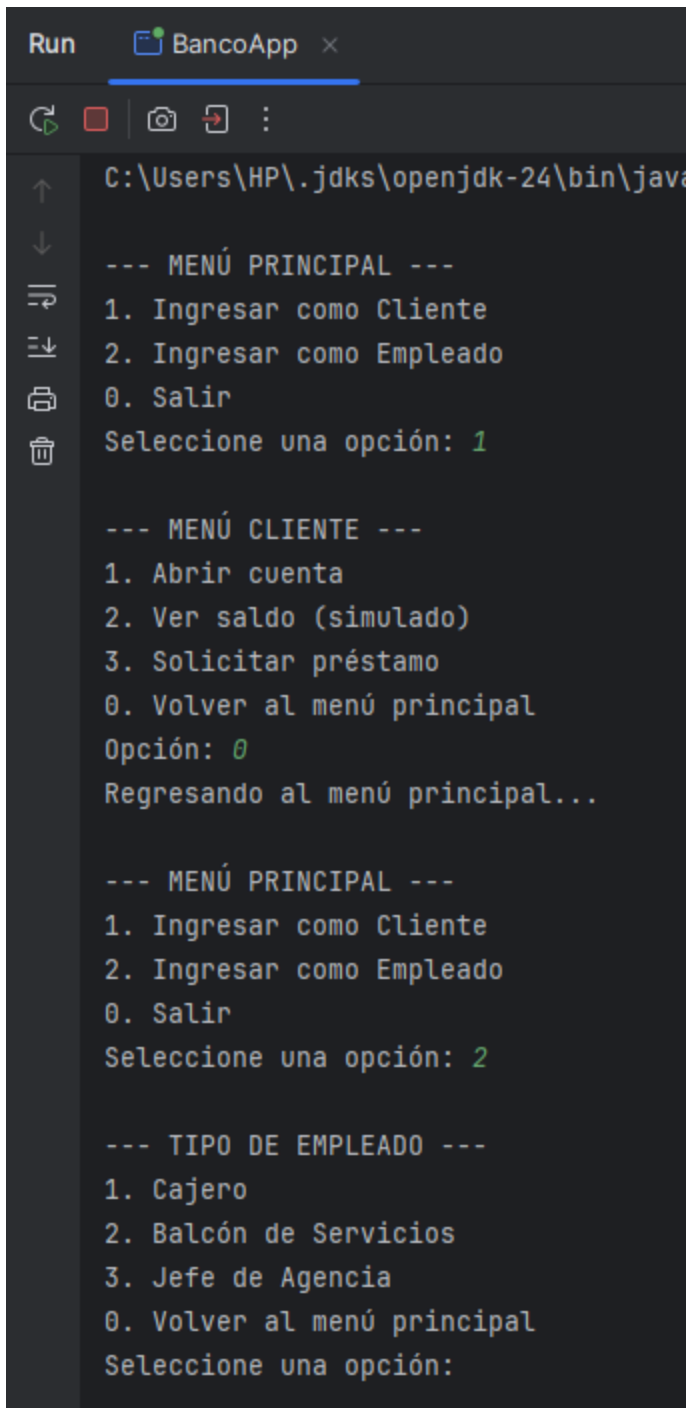


```
1 public class JefeAgencia extends Empleado { no usages
2
3     public JefeAgencia(String nombre, String cedula, String direccion, String telefono, String usuario, String clave) { no
4         super(nombre, cedula, direccion, telefono, usuario, clave);
5     }
6
7     public boolean aprobarPrestamo(Cliente cliente, double monto) { no usages
8         System.out.println("Préstamo aprobado por $" + monto);
9         return true;
10     }
11
12     public void generarReporteOperaciones() { no usages
13         System.out.println("Generando reporte de operaciones...");
14     }
15
16     public void evaluarEmpleado(Empleado empleado) { no usages
17         System.out.println("Evaluando desempeño de: " + empleado.nombre);
18     }
19
20     @Override no usages
21     public void mostrarRol() {
22         System.out.println("Soy el jefe de agencia.");
23     }
24 }
25 }
```

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)



```
Run BancoApp x
C:\Users\HP\.jdk\openjdk-24\bin\java...
--- MENÚ PRINCIPAL ---
1. Ingresar como Cliente
2. Ingresar como Empleado
0. Salir
Seleccione una opción: 1

--- MENÚ CLIENTE ---
1. Abrir cuenta
2. Ver saldo (simulado)
3. Solicitar préstamo
0. Volver al menú principal
Opción: 0
Regresando al menú principal...

--- MENÚ PRINCIPAL ---
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:
```

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

```
C:\Users\HP\.jdk\openjdk-24\bin\java.exe

--- MENÚ PRINCIPAL ---
1. Ingresar como Cliente
2. Ingresar como Empleado
0. Salir
Seleccione una opción: 1

--- MENÚ CLIENTE ---
1. Abrir cuenta
2. Ver saldo (simulado)
3. Solicitar préstamo
0. Volver al menú principal
Opción: 1
Ingrese tipo de cuenta: Corriente
Cuenta Corriente registrada.

--- MENÚ CLIENTE ---
1. Abrir cuenta
2. Ver saldo (simulado)
3. Solicitar préstamo
0. Volver al menú principal
Opción: 3
Ingrese monto del préstamo: 1000
Préstamo solicitado: $1000.0

--- MENÚ CLIENTE ---
1. Abrir cuenta
2. Ver saldo (simulado)
3. Solicitar préstamo
0. Volver al menú principal
Opción:
```

- Cajero: retiro, depósito


```

1. Ingresar como Cliente
2. Ingresar como Empleado
0. Salir
Seleccione una opción: 2

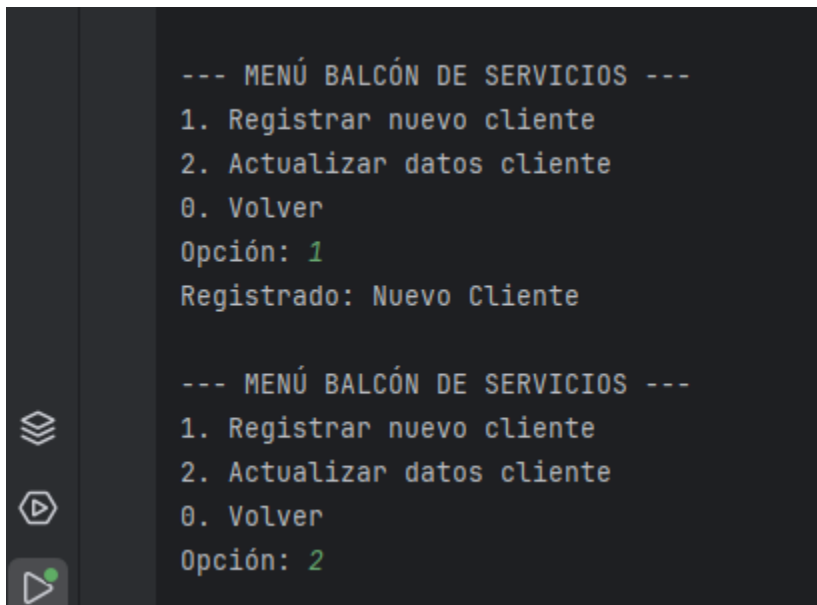
--- TIPO DE EMPLEADO ---
1. Cajero
2. Balcón de Servicios
3. Jefe de Agencia
0. Volver al menú principal
Seleccione una opción: 1

--- 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



```
--- MENÚ BALCÓN DE SERVICIOS ---  
1. Registrar nuevo cliente  
2. Actualizar datos cliente  
0. Volver  
Opción: 1  
Registrado: Nuevo Cliente  
  
--- MENÚ BALCÓN DE SERVICIOS ---  
1. Registrar nuevo cliente  
2. Actualizar datos cliente  
0. Volver  
Opción: 2
```

- Jefe de Agencia: aprobación y reportes

```
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: 3

--- MENÚ JEFE DE AGENCIA ---
1. Aprobar préstamo
2. Generar reporte de operaciones
3. Evaluar empleado
0. Volver
Opción: 1
Monto del préstamo: 1000
Préstamo aprobado por $1000.0

--- MENÚ JEFE DE AGENCIA ---
1. Aprobar préstamo
2. Generar reporte de operaciones
3. Evaluar empleado
0. Volver
Opción: 3
Evaluando desempeño de: Luis

--- MENÚ JEFE DE AGENCIA ---
1. Aprobar préstamo
2. Generar reporte de operaciones
3. Evaluar empleado
0. Volver
Opción: |
```

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 -