



# 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

```

import java.util.ArrayList;
public class Cliente extends Persona {
    private ArrayList<Cuenta> cuentas = new ArrayList<>();
    private ArrayList<TarjetaCredito> tarjetas = new ArrayList<>();

    public Cliente(String nombre, String cedula, String direccion, String telefono) {
        super(nombre, cedula, direccion, telefono);
    }

    public boolean ingresarAlSistema() {
        return true;
    }

    public void registrarCuenta(String tipo) {
        Cuenta cuenta = new Cuenta(tipo);
        cuentas.add(cuenta);
        System.out.println("Cuenta " + tipo + " registrada.");
    }

    public void solicitarPrestamo(double monto) {
        System.out.println("Préstamo solicitado: $" + monto);
    }

    public void agregarTarjetaCredito(TarjetaCredito tarjeta) {
        tarjetas.add(tarjeta);
        System.out.println("Tarjeta agregada.");
    }

    public void verResumenFinanciero() {
        System.out.println("Resumen de Cuentas:");
        for (Cuenta c : cuentas) {
            System.out.println("- " + c.getTipo());
        }
        System.out.println("Cantidad de tarjetas: " + tarjetas.size());
    }

    @Override
    public void mostrarRole() {
        System.out.println("Soy un cliente.");
    }
}

```

**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

```

public abstract class Empleado extends Persona {
    protected String usuario; 2 usages
    protected String clave; 2 usages

    public Empleado(String nombre, String cedula, String direccion, String telefono, String usuario, String clave) {
        super(nombre, cedula, direccion, telefono);
        this.usuario = usuario;
        this.clave = clave;
    }

    public boolean autenticarEmpleado(String usuario, String clave) { no usages
        return this.usuario.equals(usuario) && this.clave.equals(clave);
    }

    public void crearCuentaParaCliente(Cliente cliente, String tipo) { no usages
        cliente.registrarCuenta(tipo);
    }

    public void modificarDatosCliente(Cliente cliente) { no usages
        cliente.actualizarDatos(direccion: "Nueva dirección", telefono: "0999999999");
    }

    public void registrarPrestamo(Cliente cliente, double monto) { no usages
        cliente.solicitarPrestamo(monto);
    }

    public void cerrarCuenta(Cliente cliente, Cuenta cuenta) { no usages
        System.out.println("Cuenta cerrada.");
    }

    @Override no usages 3 implementations
    public abstract void mostrarRol();
}

```

### Métodos adicionales sugeridos por rol:

#### Para Cajero:

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

```

public class Cajero extends Empleado {
    public Cajero(String nombre, String cedula, String direccion, String telefono, String usuario, String clave) {
        super(nombre, cedula, direccion, telefono, usuario, clave);
    }

    public void procesarRetiro(Cliente cliente, double monto) { 1 usage
        System.out.println("Retiro procesado: $" + monto);
    }

    public void consultarSaldo(Cliente cliente) { 1 usage
        System.out.println("Saldo disponible: XXX");
    }

    public void procesarDeposito(Cliente cliente, double monto) { 1 usage
        System.out.println("Depósito procesado: $" + monto);
    }

    @Override no usages
    public void mostrarRol() {
        System.out.println("Soy un cajero.");
    }
}

```

#### Para BalconServicios:

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

```

public class BalconServicios extends Empleado {
    public BalconServicios(String nombre, String cedula, String direccion, String telefono, String usuario, String clave) {
        super(nombre, cedula, direccion, telefono, usuario, clave);
    }

    public Cliente registrarNuevoCliente() {
        return new Cliente( nombre: "Nuevo Cliente", cedula: "000000", direccion: "Calle X", telefono: "0988888888");
    }

    public void actualizarDatosCliente(Cliente cliente) {
        cliente.actualizarDatos( direccion: "Dirección nueva", telefono: "0999999999");
    }

    @Override no usages
    public void mostrarRole() {
        System.out.println("Soy personal de balcón de servicios.");
    }
}

```

### Para JefeAgencia:

- + aprobarPrestamo(cliente: Cliente, monto: double): boolean
- + generarReporteOperaciones(): void
- + evaluarEmpleado(empleado: Empleado): void

```

public class JefeAgencia extends Empleado {
    public JefeAgencia(String nombre, String cedula, String direccion, String telefono, String usuario, String clave) {
        super(nombre, cedula, direccion, telefono, usuario, clave);
    }

    public boolean aprobarPrestamo(Cliente cliente, double monto) {
        System.out.println("Préstamo aprobado por $" + monto);
        return true;
    }

    public void generarReporteOperaciones() {
        System.out.println("Generando reporte de operaciones...");
    }

    public void evaluarEmpleado(Empleado empleado) {
        System.out.println("Evaluando desempeño de: " + empleado.nombre);
    }

    @Override no usages
    public void mostrarRole() {
        System.out.println("Soy el jefe de agencia.");
    }
}

```

## 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 ×
C:\Users\HP\.jdks\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\.jdks\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.

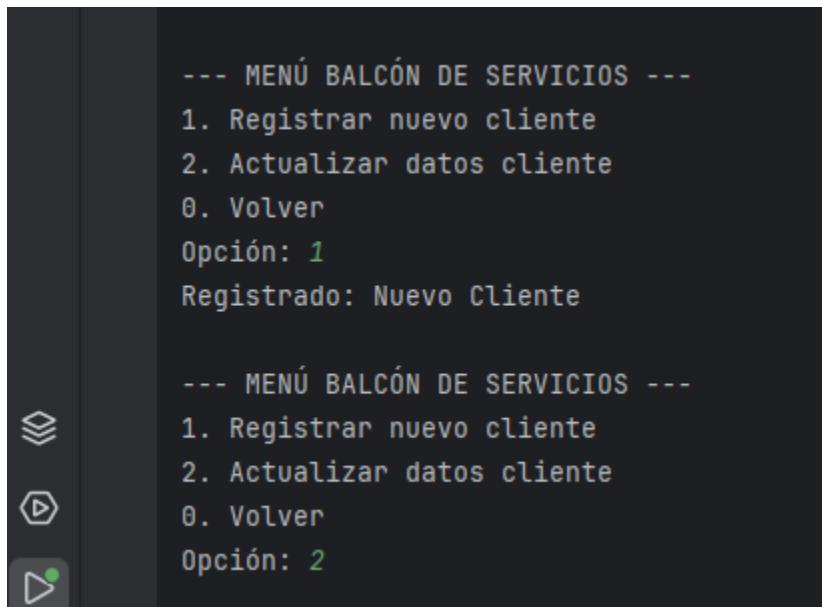
--- MENÚ CLIENTE ---
1. Abrir cuenta
2. Ver saldo (simulado)
3. Solicitar préstamo
0. Volver al menú principal
Opción: 3
Ingrese monto del préstamo: 1000
Préstamo solicitado: $1000.0

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

- Cajero: retiro, depósito

```
G □ | ☰ ☱ :  
HECHO EN PYTHON  
↑ 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** -