

**APLICANDO EL PATRON DE DISENO SINGLETON Y FACTORY**

Esta práctica cubre el patrón de diseño Singleton y Factory.

**CASO: BANCO ANTES DE APLICAR PATRONES DE DISEÑO**

Se nos muestra el siguiente código fuente para poder representar un Banco, Cuenta, Tipo de Cuenta (Cuenta Normal y Cuenta a Plazo Fijo), Cliente.

Vamos a analizar cada clase:

**Banco.java**

Esta clase es la que representa a la institución financiera. Esta posee clientes y tiene la operación de agregar clientes, obtener número de clientes o un cliente dado el índice.

|  |
| --- |
| import java.util.ArrayList;  import java.util.List;  public class Banco {  private List<Cliente> clientes;  public Banco() {  clientes = new ArrayList<>();  }  public void agregarCliente(Cliente cliente) {  clientes.add(cliente);  }  public List<Cliente> getClientes() {  return clientes;  }  public void setClientes(List<Cliente> clientes) {  this.clientes = clientes;  }    } |

**Cliente.java**

Esta clase Cliente representa a los clientes del Banco. En esta clase tendremos información de cliente cómo: Nombres, Apellidos, Cuentas, numero de Cuentas.

|  |
| --- |
| import java.util.ArrayList;  import java.util.List;  public class Cliente {  private String nombres;  private String apellidos;  private List<Cuenta> cuentas;    public Cliente(String nombres, String apellidos) {  this.nombres = nombres;  this.apellidos = apellidos;  // initializar array de cuentas  cuentas = new ArrayList<Cuenta>();  }  public String getNombres() {  return nombres;  }  public String getApellidos() {  return apellidos;  }  public void agregarCuenta(Cuenta cuenta) {  cuentas.add(cuenta);  }  public List<Cuenta> getCuentas() {  return cuentas;  }  public void setCuentas(List<Cuenta> cuentas) {  this.cuentas = cuentas;  }  } |

**Cuenta.java**

Esta clase representa a una cuenta del cliente. Tiene como información el balance, operaciones como depositar, retirar, obtener descripción de la cuenta.

Se declara abstracta para que tanto **CuentaNormal** y **CuentaAPlazoFijo** hereden de ella e implementen el método **retirar** y **getDescripcion** que depende de las clases concretas.

|  |
| --- |
| public abstract class Cuenta {    protected double balance;    public Cuenta(double balance) {  this.balance = balance;  }    public double getBalance() {  return balance;  }    public void depositar(double cantidad) {  balance += cantidad;  }    public String toString() {  return getDescripcion() + ": El actual balance es " + balance;  }    public abstract boolean retirar(double cantidad);    public abstract String getDescripcion();    } |

**CuentaNormal.java**

Este tipo de cuenta representa a una cuenta norma donde se puede depositar y retirar. Se puede tener un crédito máximo si la institución lo permite.

|  |
| --- |
| public class CuentaNormal extends Cuenta{  private final double creditoMaximo;    public CuentaNormal(double balance) {  this(balance, 0);  }    public CuentaNormal(double balance, double creditoMaximo) {  super(balance);  this.creditoMaximo = creditoMaximo;  }    @Override  public boolean retirar(double cantidad) {  if(cantidad <= balance + creditoMaximo) {  balance -= cantidad;  return true;  } else {  return false;  }  }  @Override  public String getDescripcion() {  return "Cuenta Normal";  }    } |

**CuentaAPlazoFijo.java**

Este tipo de cuenta representa a una cuenta a plazo fijo. Esta cuenta tiene una fecha de vencimiento y solo se puede retirar después de que ha vencido dicha fecha.

|  |
| --- |
| import java.util.Date;  public class CuentaAPlazoFijo extends Cuenta{  private final Date fechaVencimiento;    public CuentaAPlazoFijo(double balance, Date fechaVencimiento) {  super(balance);  this.fechaVencimiento = fechaVencimiento;  }  @Override  public boolean retirar(double cantidad) {    Date hoy = new Date();  if(hoy.after(fechaVencimiento)) {  if(cantidad <= balance) {  balance -= cantidad;  return true;  } else {  return false;  }  } else {  return false;  }  }  @Override  public String getDescripcion() {  return "Cuenta a plazo fijo con fecha de vencimiento " +fechaVencimiento;  }  } |

**ReporteCliente.java**

Esta clase es para mostrar información de los clientes del banco y por cada cliente el balance de sus cuentas.

|  |
| --- |
| public class ReporteCliente {  private Banco banco;  public Banco getBanco() {  return banco;  }  public void setBanco(Banco banco) {  this.banco = banco;  }  public void generarReporte() {  // Imprimir cabecera del reporte  System.out.println("\t\t\tREPORTE DE CLIENTES");  System.out.println("\t\t\t================");  // Para cada cliente  for(Cliente p : this.banco.getClientes()){  // Imprimir el nombre del cliente  System.out.println();  System.out.println("Cliente: "  + p.getApellidos()+ ", "  + p.getNombres());  // Para cada cuenta de este cliente  for (Cuenta c: p.getCuentas() ) {  // Imprimir el balance actual de la cuenta  System.out.println(" " + c);  }  }  }  } |

**ActividadBanco.java**

Aquí el método main permite crear un Banco, inicializar clientes, cuentas y ver el reporte de clientes.

|  |
| --- |
| package actividadbanco;  import java.util.Calendar;  public class ActividadBanco {  public static void main(String[] args) {  Banco banco = new Banco();  inicializarClientes(banco);  // Ejecutar el reporte de cliente  ReporteCliente reporte = new ReporteCliente();  reporte.setBanco(banco);  reporte.generarReporte();  }    private static void inicializarClientes(Banco banco) {  //A 180 dias  Calendar cal = Calendar.getInstance();  cal.add(Calendar.DAY\_OF\_YEAR, 180);    Cliente cliente1 = new Cliente("Jose", "Diaz");  // Creamos diferentes clientes y sus cuentas  banco.agregarCliente(cliente1);  cliente1.agregarCuenta(new CuentaAPlazoFijo(500.00, cal.getTime()));  cliente1.agregarCuenta(new CuentaNormal(200.00, 400.00));    Cliente cliente2 = new Cliente("Armando", "Lopez");  banco.agregarCliente(cliente2);  cliente2.agregarCuenta(new CuentaNormal(200.00));  cliente2.agregarCuenta(new CuentaAPlazoFijo(1500.00, cal.getTime()));  Cliente cliente3 = new Cliente("Ana", "Loo");    banco.agregarCliente(cliente3);  cliente3.agregarCuenta(new CuentaNormal(200.00));  cliente3.agregarCuenta(new CuentaAPlazoFijo(150.00, cal.getTime()));  }  } |

**ACTIVIDAD A REALIZAR**

1. Ud. Ha revisado el patrón Singleton en clase. **¿A cuál de estas clases considera Ud. se debe aplicar el Patrón Singleton?**. Justifique y modifique el proyecto adjunto (Banco.zip) que contiene el código fuente.

El nuevo proyecto deberá dar el mismo resultado pero esta vez usando el patrón de diseño Singleton. (Comente todas las modificaciones que realice usando //).

1. Realice una segunda modificación a partir del último código modificado y aplique el patrón de diseño Factory simple, de tal manera que esta factoría genere objetos de las clases **CuentaNormal** y **CuentaAPlazoFijo**.

El nuevo proyecto deberá dar el mismo resultado pero esta vez usando el patrón de diseño Singleton + Factory. (Comente todas las modificaciones que realice usando //).

1. Envíe el proyecto Banco.zip generado, al finalizar las modificaciones.