Universidad Técnica Particular de Loja

Computación

Programación Orientada a Objetos

Evelyn Lizzethe Riofrío Riofrío
David Alejandro Mora Hidalgo

Proyecto Bimestral 2B

Abril 2025 - agosto 2025

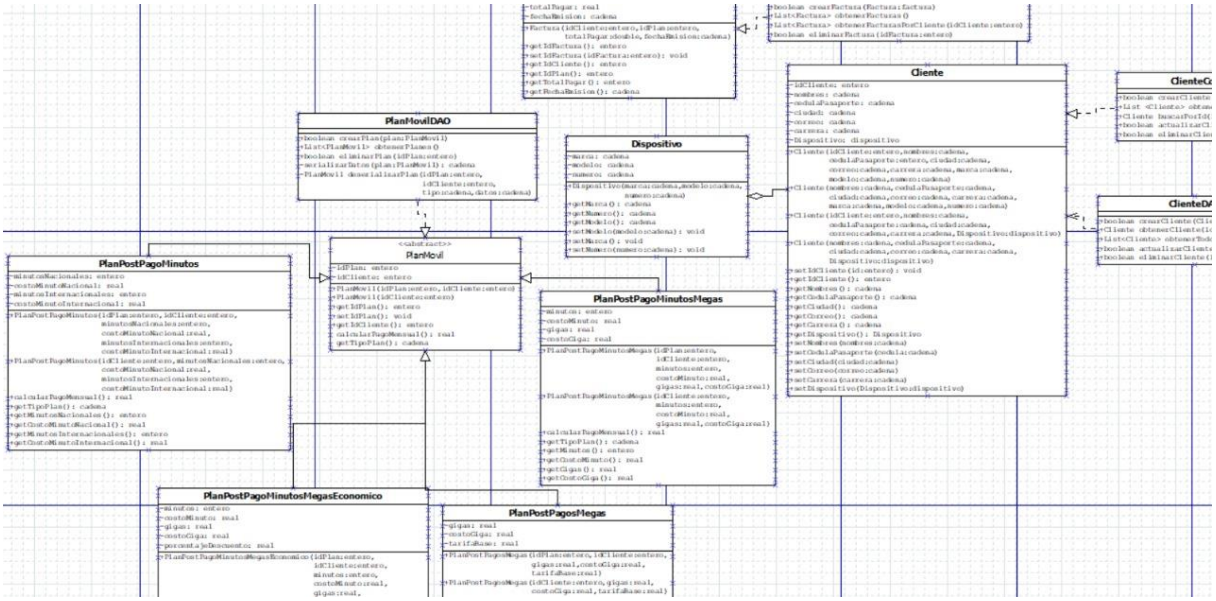
Loja – Ecuador

Análisis del problema

Problema Detectado	Falta de un sistema automatizado para gestionar clientes, planes móviles y facturación en Mov-UTPL.
Objetivo General	Diseñar e implementar un sistema en Java (MVC) con SQLite que permita CRUD de clientes y planes, y generación de facturas aplicando POO (herencia y polimorfismo).
Datos de Entrada	• Datos de clientes: nombres, cédula/pasaporte, ciudad, marca/modelo celular, número celular, correo, fechaRegistro. • Datos de planes: según el tipo de plan (minutos, megas, costos, descuentos).
Procesos	1. Registro CRUD de clientes. 2. Registro CRUD de planes móviles. 3. Asociación cliente-plan. 4. Cálculo automático de facturación según plan. 5. Persistencia en SQLite.
Salidas	- Listado de clientes y planes. - Facturas detalladas por cliente. - Reportes de pagos mensuales.
POO aplicada	- Herencia: Clase base Plan con subclases especializadas. - Polimorfismo: Método calcularPagoMensual() sobrescrito por cada tipo de plan. - Encapsulamiento: Uso de getters/setters. - Abstracción: Separación de responsabilidades mediante MVC.
Base de Datos (SQLite)	Tablas: 1. clientes (PK: cedula) 2. planes (PK: idPlan, FK: cedulaCliente) 3. facturas (PK: idFactura, FK: cedulaCliente, FK: idPlan).
Arquitectura MVC	- Modelo: Clases Cliente, Plan y subclases, Factura, y DAOs. - Vista: Interfaz de usuario (Swing y consola). - Controlador: Coordinación entre vista y modelo.

Beneficios Esperados	- Automatización del registro y facturación. - Reducción de errores manuales. - Eficiencia en el control de planes y pagos.

Modelado UML:



Codificación:

Creación clase cliente con constructor que sigue la estructura de la base de datos:

```
public class Cliente {

    private int idCliente;
    private String nombres;
    private String cedulaPasaporte;
    private String ciudad;
    private String correo;
    private String carrera;
    private Dispositivo dispositivo;

    // Constructor principal (lectura desde BD)
    public Cliente(int idCliente,
            String nombres,
            String cedulaPasaporte,
            String ciudad,
            String correo,
            String carrera,
            String marca,
            String modelo,
            String numero) {
        this.idCliente = idCliente;
        this.nombres = nombres;
        this.cedulaPasaporte = cedulaPasaporte;
        this.ciudad = ciudad;
        this.correo = correo;
        this.carrera = carrera;
        this.dispositivo = new Dispositivo(marca, modelo, numero);
```

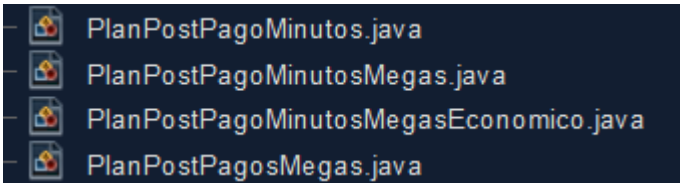
Así mismo, un constructor que no requiere ID (Con este podremos generarlo automáticamente al momento de ejecutar la vista) Y un constructor que acepte directamente un dispositivo:

```java
// Para crear antes de persistir (id=0)
public Cliente(String nombres,
        String cedulaPasaporte,
        String ciudad,
        String correo,
        String carrera,
        String marca,
        String modelo,
        String numero) {
    this(0, nombres, cedulaPasaporte, ciudad, correo, carrera, marca, modelo, numero);
}

// Este constructor acepta directamente un Dispositivo (puede ser null)
public Cliente(int idCliente,
        String nombres,
        String cedulaPasaporte,
        String ciudad,
        String correo,
        String carrera,
        Dispositivo dispositivo) {
    this.idCliente = idCliente;
    this.nombres = nombres;
    this.cedulaPasaporte = cedulaPasaporte;
    this.ciudad = ciudad;
    this.correo = correo;
    this.carrera = carrera;
    this.dispositivo = dispositivo;  // Asignación directa, sin llamar a getters
}
```

Clase Abstracta PlanMovil que recibe el id del cliente y el dispositivo para así poder asociarlos y finalmente sacar un cálculo en base a eso(con sus métodos abstractos):

```java
package Modelo;

public abstract class PlanMovil {

    private int idPlan;
    private int idCliente;

public abstract double calcularPagoMensual();

public abstract String getTipoPlan();
```

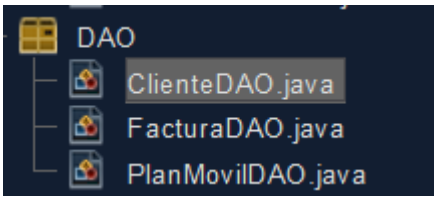
Teniendo de clases hijas las siguientes, cada una con su respectiva lógica de cálculo y atributos propios:



Como clase final dentro del Modelo, tenemos a la factura, que como podemos observar en sus atributos, asocia el id del cliente con el id del plan para poder generar una nueva tabla de resultados en base a la información previamente recolectada:

```java
package Modelo;

public class Factura {

    private int idFactura;
    private int idCliente;
    private int idPlan;
    private double totalPagar;
    private String fechaEmision;

    public Factura(int idCliente, int idPlan, double totalPagar, String fechaEmision) {
        this.idCliente = idCliente;
        this.idPlan = idPlan;
        this.totalPagar = totalPagar;
        this.fechaEmision = fechaEmision;
```

Siguiente, creamos cada clase DAO, que serán la que nos permitan acceder a las bases de datos e implementar los métodos CRUD:



Empezamos por Cliente DAO(revisaremos los puntos importantes), que, como novedad, no nos pide ingresar un id, si no que lo genera automáticamente (por esto la creación del segundo constructor revisador anteriormente):

```java
public boolean crearCliente(Cliente cliente) {
    String sql = """
  INSERT INTO clientes
    (nombres, cedula_pasaporte, ciudad, correo, carrera, marca, modelo, numero)
  VALUES (?, ?, ?, ?, ?, ?, ?, ?)""";

    try (Connection conn = ConexionSQLite.conectar(); //  Pedimos que nos devuelva el ID autogenerado:
            PreparedStatement pstmt = conn.prepareStatement(sql, Statement.RETURN_GENERATED_KEYS)) {

        //  Aquí _sí_ vinculamos **todos** los parámetros:
        pstmt.setString(1, cliente.getNombres());
        pstmt.setString(2, cliente.getCedulaPasaporte());
        pstmt.setString(3, cliente.getCiudad());
        pstmt.setString(4, cliente.getCorreo());
        pstmt.setString(5, cliente.getCarrera());

        Dispositivo d = cliente.getDispositivo();
        if (d != null) {
            pstmt.setString(6, d.getMarca());
            pstmt.setString(7, d.getModelo());
            pstmt.setString(8, d.getNumero());
        } else {
            pstmt.setNull(6, Types.VARCHAR);
            pstmt.setNull(7, Types.VARCHAR);
            pstmt.setNull(8, Types.VARCHAR);
        }

        //  Ejecutamos la inserción:
        int filas = pstmt.executeUpdate();
        if (filas == 0) {
            return false;  // no se insertó nada
        }
```

La función que nos permite obtener el cliente en base a su ID:

```
public Cliente obtenerClientePorId(int id) {
    String sql = "SELECT * FROM clientes WHERE id_cliente = ?";
    try (Connection conn = ConexionSQLite.conectar(); PreparedStatement pstmt = conn.prepareStatement(sql)) {

        pstmt.setInt(1, id);
        try (ResultSet rs = pstmt.executeQuery()) {
            if (rs.next()) {
                return new Cliente(
                        rs.getInt("id_cliente"),
                        rs.getString("nombres"),
                        rs.getString("cedula_pasaporte"),
                        rs.getString("ciudad"),
                        rs.getString("correo"),
                        rs.getString("carrera"),
                        rs.getString("marca"),
                        rs.getString("modelo"),
                        rs.getString("numero")
                );
            }
        }
```

Siguiente, tenemos el DAO del Plan Móvil, que aparte de sus métodos de inserción, actualización y eliminación, tiene los de serializar y de-serializar los datos:

```
private String serializarDatos(PlanMovil plan) {
    if (plan instanceof PlanPostPagoMinutos) {
        PlanPostPagoMinutos p = (PlanPostPagoMinutos) plan;
        return p.getMinutosNacionales() + ";" +
                p.getCostoMinutoNacional() + ";" +
                p.getMinutosInternacionales() + ";" +
                p.getCostoMinutoInternacional();
    } else if (plan instanceof PlanPostPagoMinutosMegasEconomico) {
        PlanPostPagoMinutosMegasEconomico p = (PlanPostPagoMinutosMegasEconomico) plan;
        return p.getMinutos() + ";" +
                p.getCostoMinuto() + ";" +
                p.getGigas() + ";" +
                p.getCostoGiga() + ";" +
                p.getPorcentajeDescuento();
    } else if (plan instanceof PlanPostPagoMinutosMegas) {
        PlanPostPagoMinutosMegas p = (PlanPostPagoMinutosMegas) plan;
        return p.getMinutos() + ";" +
                p.getCostoMinuto() + ";" +
                p.getGigas() + ";" +
                p.getCostoGiga();
    } else if (plan instanceof PlanPostPagosMegas) {
        PlanPostPagosMegas p = (PlanPostPagosMegas) plan;
        return p.getGigas() + ";" +
                p.getCostoGiga() + ";" +
                p.getTarifaBase();
    }
    return "";
}
```

```
private PlanMovil deserializarPlan(int idPlan, int idCliente, String tipo, String datos) {
    String[] p = datos.split(";");
    try {
        if ("PostPagoMinutos".equals(tipo)) {
            return new PlanPostPagoMinutos(
                idPlan,
                idCliente,
                Integer.parseInt(p[0]),
                Double.parseDouble(p[1]),
                Integer.parseInt(p[2]),
                Double.parseDouble(p[3])
            );
        } else if ("PostPagoMinutosMegasEconomico".equals(tipo)) {
            return new PlanPostPagoMinutosMegasEconomico(
                idPlan,
                idCliente,
                Integer.parseInt(p[0]),
                Double.parseDouble(p[1]),
                Double.parseDouble(p[2]),
                Double.parseDouble(p[3]),
                Double.parseDouble(p[4])
            );
        } else if ("PostPagoMinutosMegas".equals(tipo)) {
            return new PlanPostPagoMinutosMegas(
                idPlan,
                idCliente,
                Integer.parseInt(p[0]),
                Double.parseDouble(p[1]),
                Double.parseDouble(p[2]),
                Double.parseDouble(p[3])
            );
        } else if ("PostPagoMegas".equals(tipo)) {
            return new PlanPostPagosMegas(
                idPlan,
                idCliente,
```

Esto se hace con la finalidad de que al momento de ingresar los datos del plan como lo son los minutos, costos, etc, al serializarlos se puedan guardar como solo una cadena de texto que se almacene en la columna de datos del plan, lo cual nos evita la necesidad de crear una tabla nueva para cada tipo de plan. Y el proceso de deserialización se realiza para que podamos trabajar con esos datos nuevamente como objetos y poder realizar todos los cálculos asociados a ello.

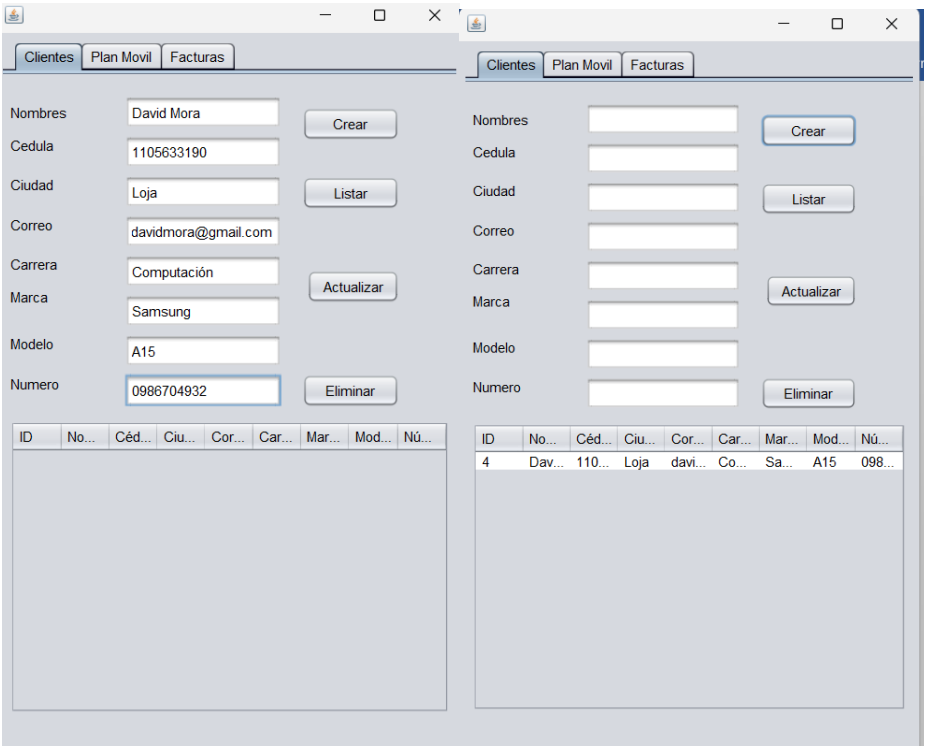
Finalmente tenemos el DAO de la factura que calcula el monto final en base a la información almacenada por el id del cliente y el id del plan, como podemos observar aquí:

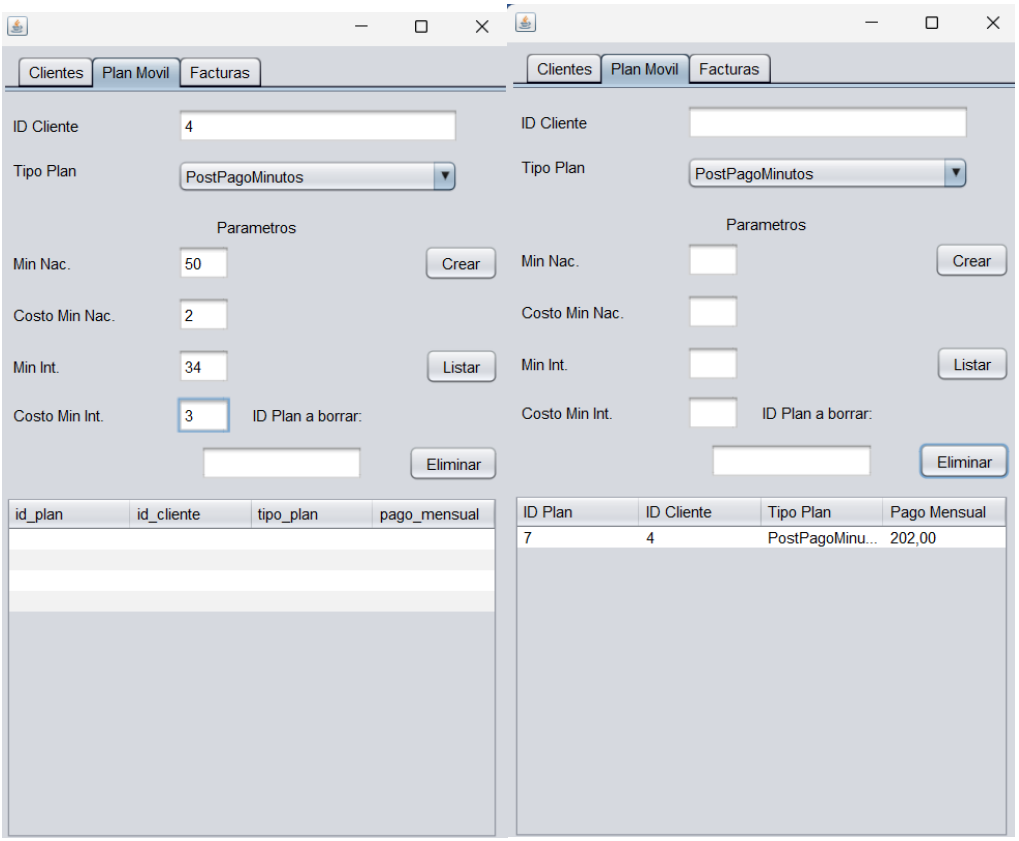
```
public class FacturaDAO {

    public boolean crearFactura(Factura factura) {
        String sql = """
            INSERT INTO facturas
              (id_cliente, id_plan, total_pagar, fecha_emision)
            VALUES (?, ?, ?, ?)
            """;
        try (Connection conn = ConexionSQLite.conectar(); PreparedStatement pstmt = conn.prepareStatement(sql)) {

            pstmt.setInt(1, factura.getIdCliente());
            pstmt.setInt(2, factura.getIdPlan());
            pstmt.setDouble(3, factura.getTotalPagar());
            pstmt.setString(4, factura.getFechaEmision());  // ya es String

            int filas = pstmt.executeUpdate();
            if (filas == 0) {
                return false;
            }

            try (Statement st = conn.createStatement(); ResultSet rs = st.executeQuery("SELECT last_insert_rowid()")) {
                if (rs.next()) {
                    factura.setIdFactura(rs.getInt(1));
                }
            }
```

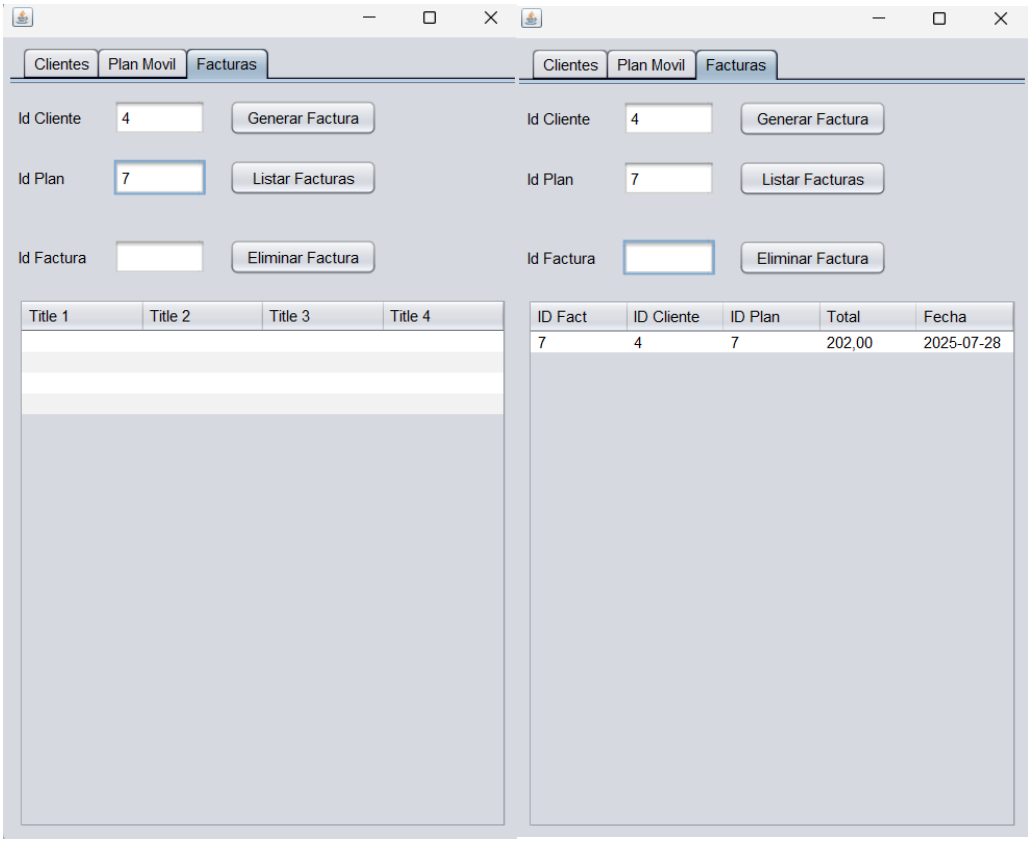
Dándonos nuestra tabla final con la factura emitida.

Resultados:

Finalmente ejecutamos nuestra interfaz gráfica, mediante la cual podremos ingresar los datos y recibir nuestros resultados:






Clientes
Plan Movil
Facturas
Id Cliente
4
Generar Factura
Id Plan
7
Listar Facturas
Id Factura
Eliminar Factura
Title 1
Title 2
Title 3
Title 4
Clientes
Plan Movil
Facturas
Id Cliente
4
Generar Factura
Id Plan
7
Listar Facturas
Id Factura
Eliminar Factura
ID Fact
ID Cliente
ID Plan
Total
Fecha
7
4
7
202,00
2025-07-28