UNIVERSIDAD TECNICA PARTICULAR DE LOJA
La Universidad Católica de Loja

FACULTAD: Ingenierías y Arquitectura

CARRERA: Ing. en ciencias de la computación

ASIGNATURA: Programación Orientada a Objetos

TEMA: Gestión de telefonía móvil estudiantil: Mov-UTPL.

ESTUDIANTES:

- Danny Guachisaca
- Santiago Villacorta

DOCENTE: Ing. Wayner Bustamante

CICLO: 2 Ciclo
FECHA DE ENTREGA: 30 de julio de 2025

Loja – Ecuador

Índice

1. Introducción

En este breve informe vamos a abarcar la descripción total de nuestro proyecto de investigación y aplicación de los conceptos aprendidos durante este segundo bimestre, realizaremos un sistema de gestión de administración de facturas y clientes, aplicando conceptos básicos como programación en java, la arquitectura MVC usando bases de datos de tipo SQLite. Este proyecto de final de bimestre se ha hecho como parte de la notal final de la asignatura programación de objetos.

Este sistema de gestión de planes telefónicos, busca como principal objetivo permitirle al administrador un fácil registro de los datos del cliente, la gestión de cada tipo de plan contratado y una generación de facturas de fácil comprensión al lector, como también aplicar buenas prácticas de programación.

Durante toda la codificación del programa hemos usado diferentes tipos de herramientas, tales como NetBeans que fue de gran ayuda que fue la principal IDE de programación utilizada, la app DB browser (SQLite) la cual fue óptima para la creación de base de datos y Junit 4 para poder hacer pruebas de corrida del programa. Por último, usamos Dia para poder realizar diagramas UML que fueron punto clave para evidenciar la lógica del programa.

2. Análisis del problema

2.1 Descripción de la problemática

En la actualidad, varias instituciones tanto universitarias como de segundo nivel tienen dificultades relacionadas al control de recepción de datos de los servicios de telefonía móvil de la comunidad estudiantil. Esto hace referencia a los datos generales del estudiante (Nombre, edad, etc.), el tipo de plan móvil, el control de pagos y un sistema de facturación de fácil comprensión.

Este sistema busca poder agilizar la entrada de datos mediante una aplicación en Java, que permita al personal educativo registrar estudiantes, el tipo de plan pertinente para el tipo de estudiante, el control de pagos y la entrega de facturas. Esto quita contratiempos al personal educativo ya que hacer todo este proceso de manera Manuel es muy tedioso y puede generar errores o confusiones futuras.

2.2 Personas y atributos

Persona	**Atributos**
Cliente	Cedula, nombre, correo, teléfono

Plan Móvil	Marca, modelo, numero, pago
Factura	Id, cliente id, plan id, total, fecha emisión

2.3 Relación entre personas

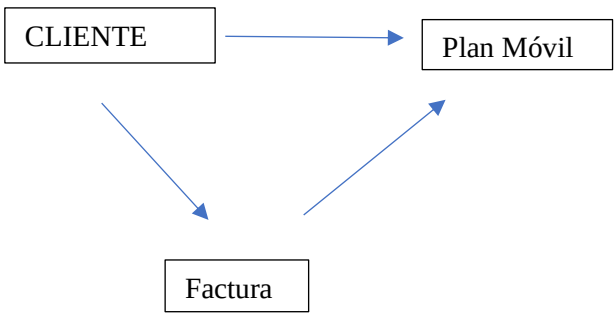
a. La clase cliente como representa a un estudiante debe tener asignado un Plan móvil
b. La clase Factura tiene una relación directa con Cliente y Plan móvil
c. Las relaciones SQL, usan de manera contante JOINs para poder unir los datos de varias tablas y de esta manera poder obtener los datos completos para poder emitir una factura.

2.4 Justificación de Uso

Con mi grupo decidimos usar bases de datos de tipo SQLite por su facilidad de creación de tablas, su portabilidad y su uso con java, la cual nos ayudo a ejemplifica los siguientes puntos:

a. Nos ayuda a evitar doble entrada de datos o la duplicación de los mismos ya que separa las tablas para cada tipo de base (cliente, plan, factura)
b. Hay relación entre las bases de datos para tomar los valores de cada una de estas, asignando valores necesarios para emitir la factura
c. Podemos acceder a la diversa información entre tablas mediante el uso de JOINs
d. Hay una consistencia y fidelidad en los datos

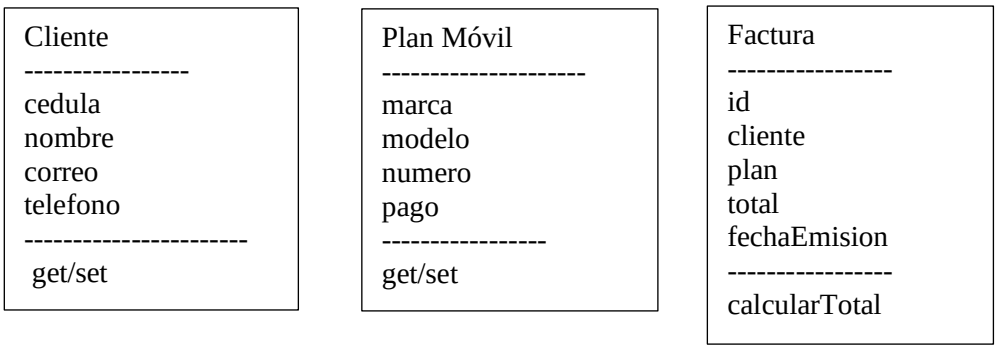
2.5 Esquema Simple



3. Diseño de algoritmo

Para realizar la solución de esta problemática creamos clases en paquetes específicos que respetan los principios solid, Creamos tres bases de datos de tipos SQL, llamadas Cliente, Plan Móvil y factura.

3.1 Esquema Simple



3.2 Salida al Usuario

3.3 Aplicación de los principios solid

SRP: Las clases se separaron las clases para que cada una tenga una responsabilidad propia, tal como
OCP: El sistema nos permite añadir nuevos tipos de planes telefónicos sin cambiarla lógica general del cliente
LSP: Se pueden variar los objetos de la clase Plan móvil sin que cambie el funcionamiento del programa
ISP: Cada clase tiene métodos independientes, usados únicamente por es clase
DIP: Las dependencias individuales se manejan desde el controlador, no desde la clase vista

4. Codificación

4.1 Fragmentos mas importantes

En este fragmento es una representación de como se valida y sew registran los clientes en la base de datos:

```
// Crear un cliente
public boolean crearCliente(Cliente cliente) {
   if (cliente == null || cliente.getIdentificacion() == null ||
cliente.getIdentificacion().isEmpty()) {
      System.out.println("Cliente inválido o identificación vacía.");
      return false;
   }
   Cliente existe = clienteDAO.buscarPorIdentificacion(cliente.getIdentificacion());
   if (existe != null) {
      System.out.println("Ya existe un cliente con esa identificación.");
      return false;
   }
   clienteDAO.insertar(cliente);
   return true;
}
```

manera directa el total a pagar cuando se crea una factura:

```
public Factura(int numFactura, String cliente, PlanCelular plan, String fechaEmision) {
    this.numFactura = numFactura;
    this.cliente = cliente;
    this.plan = plan;
    this.fechaEmision = fechaEmision;
    calcularTotal(); // Calculamos el total automáticamente al crear la factura
}

public void calcularTotal() {
    if (plan != null) {
        plan.calcularPagoMensual();  // Asegura que esté actualizado
        this.totalAPagar = plan.getPagoMensual();
    }
}
```

En este fragmento de código se inserta un cliente en la base de datos SQLite

```
public void insertar(Cliente c) {
    String sql = "INSERT INTO Cliente (identificacion, nombre, ciudad, marca, modelo,
numeroCelular, pagoMensual, fechaRegistro) VALUES (?, ?, ?, ?, ?, ?, ?, ?)";
    try (Connection conn = ConexionSQLite.conectar(); PreparedStatement ps =
conn.prepareStatement(sql)) {
        ps.setString(1, c.getIdentificacion());
        ps.setString(2, c.getNombre());
        ps.setString(3, c.getCiudad());
        ps.setString(4, c.getMarca());
        ps.setString(5, c.getModelo());
        ps.setString(6, c.getNumeroCelular());
        ps.setFloat(7, c.getPagoMensual());
        ps.setString(8, c.getFechaRegistro());
```

En este fragmento de código de la clase FacturaDAO, reconstruye un Plan celular

```
private PlanCelular crearPlanPorTipo(String tipoPlan) {
    switch (tipoPlan) {
        case "PlanPostPagoMegas":
            return new PlanPostPagoMegas();
        case "PlanPostPagoMinutos":
            return new PlanPostPagoMinutos();
        case "PlanPostPagoMinutosMegas":
            return new PlanPostPagoMinutosMegas();
        case "PlanPostPagoMinutosMegasEconomico":
            return new PlanPostPagoMinutosMegasEconomico();
        default:
            System.err.println("Tipo de plan desconocido: " + tipoPlan);
            return null;
    }
}
```

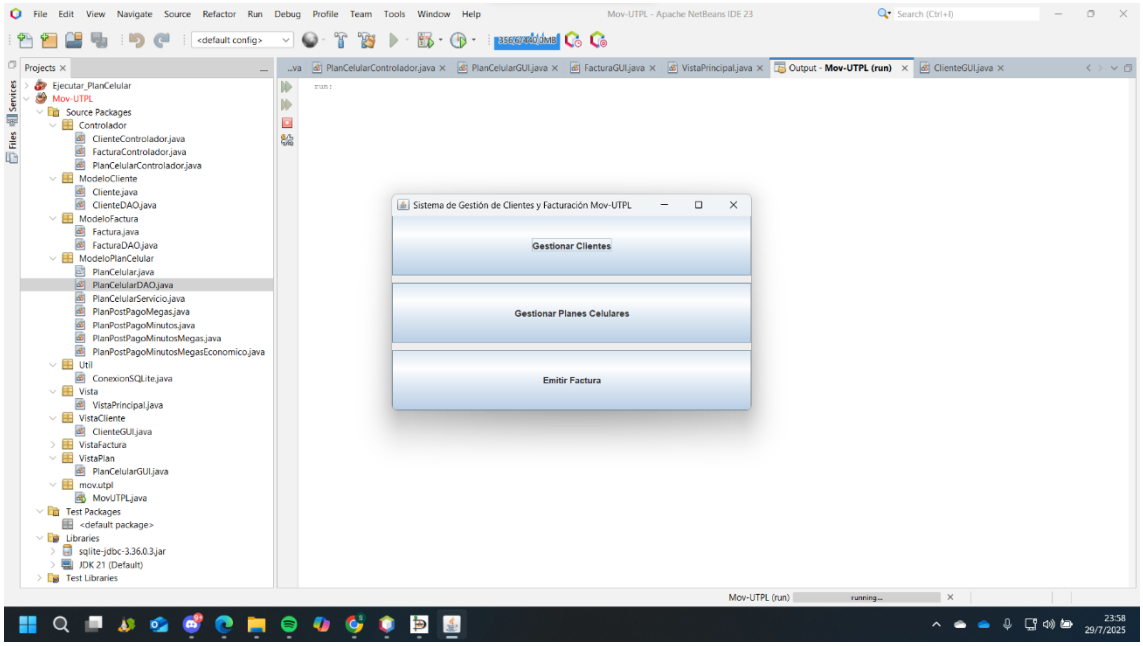
usando los datos guardados:

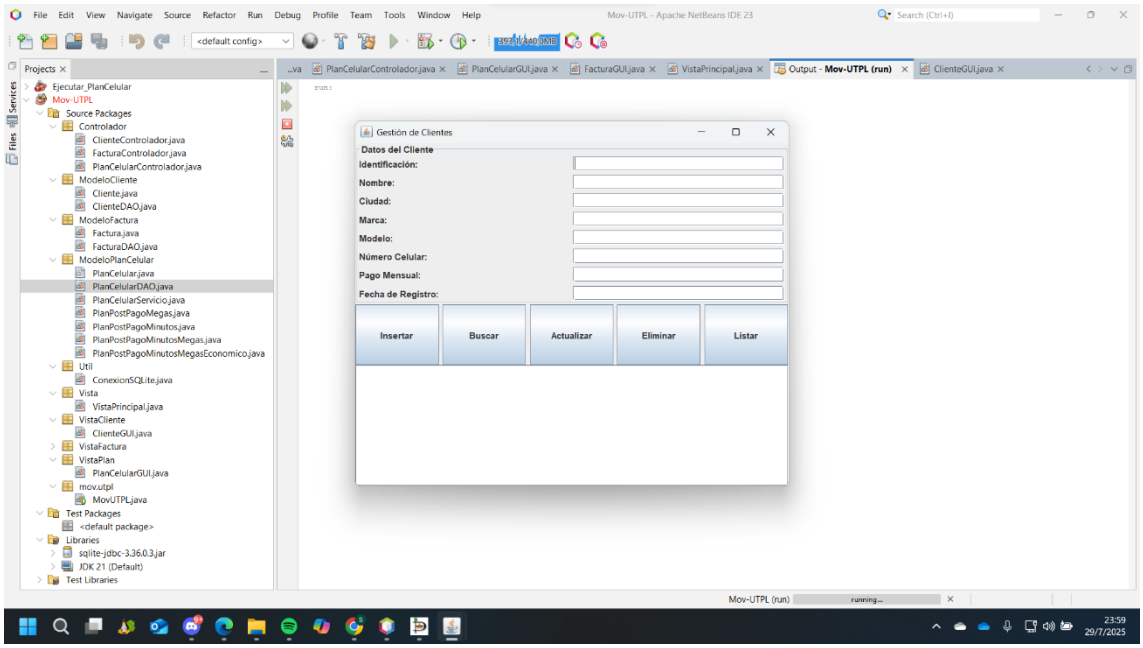
5. Repositorio con el código

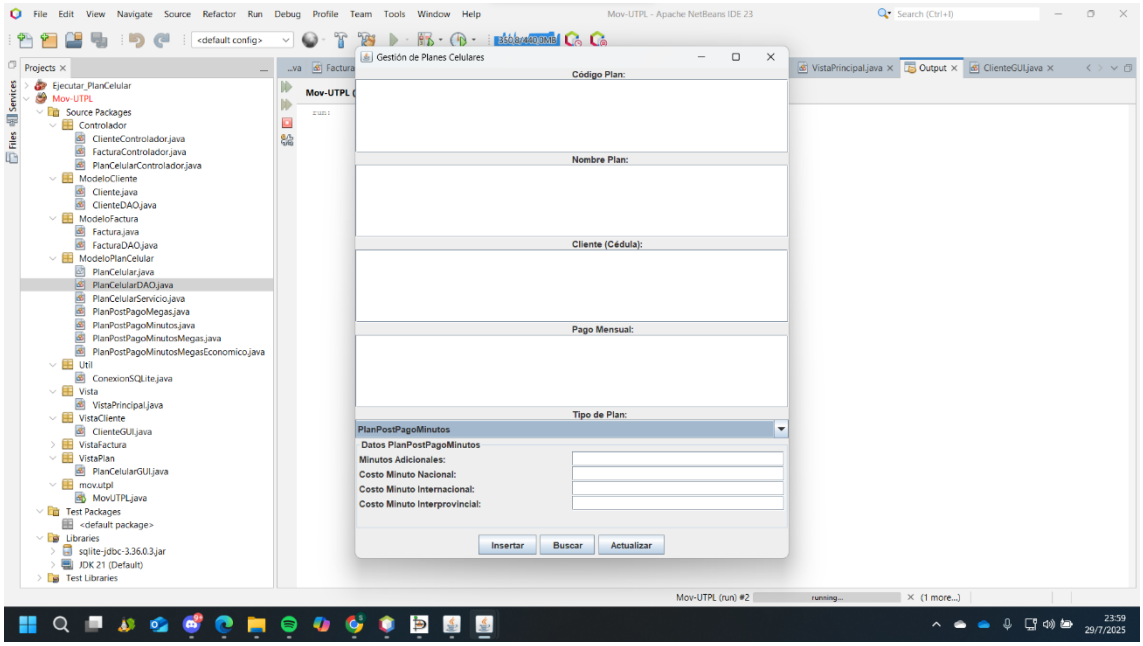
https://github.com/Santiagoc4rk/AA-TrabFinal-POO_2B.git

6. Resultados

6.1 Capturas del sistema de gesti







7. Conclusiones

Este sistema nos ayudó agilizar la gestión de la telefonía móvil en un ambiente educativo y así aprovechar de mejor manera los recursos y el personal administrativo, como también el uso de MVC y de los principios SOLID nos ayudó a mejorar la manera en la que organizamos las clases y así tener una mayor mantenibilidad del código y SQLite nos ayudo a entender de mejor manera el aprovechamiento y el correcto uso de las bases de datos, por consiguiente aprendimos que las consultas JOINs son importantes para comprobar la factibilidad de proyectos arraigados a bases de datos

8. Recomendaciones

Seria recomendable para los futuros programadores que van a basarse en este código agregar validaciones al momento de el ingreso de credenciales de un alumno ya que son valores que necesitan protección.

9. Bibliografía

a) **Elmasri, R., & Navathe, S. B.** (2011).
Sistemas de Bases de Datos (6ª ed.). Pearson Educación.
→ Explica los conceptos de modelos relacionales, SQL y tipos de JOIN de forma detallada.

b) **Silberschatz, A., Korth, H. F., & Sudarshan, S.** (2013).
Fundamentos de Bases de Datos (6ª ed.). McGraw-Hill.
→ Incluye teoría sobre consultas relacionales y operadores JOIN en SQL.

c) **W3Schools en español**.
Tutorial SQL JOIN. Recuperado de: https://www.w3schools.com/sql/sql_join.asp
→ Guía sencilla y práctica con ejemplos básicos de INNER JOIN, LEFT JOIN, etc.