Universidad Técnica Particular de Loja

Programación Orientada a Objetos

Proyecto Bimestral Grupal

Sistema de Gestión de clientes y facturación Mov-UTPL

Nombres:

Santiago Fernando Rosales Vivanco

María Valentina Samaniego Vásquez

Docente:

Wayner Xavier Bustamante Granda

Paralelo:

“D”

Análisis de la solución:

El programa desarrollado tiene como objetivo gestionar la telefonía móvil estudiantil de la Universidad Técnica Particular de Loja (UTPL), utilizando principios de programación orientada a objetos (POO). El sistema implementa herencia y polimorfismo para manejar diferentes tipos de planes postpago, y gestiona la interacción con una base de datos SQLite a través del patrón de diseño Modelo-Vista-Controlador (MVC).

Estructura del Sistema
El sistema está organizado en varios paquetes:

Controller: Contiene las clases que manejan la lógica del negocio y la interacción con la base de datos.

- **GestionClientes:** Gestiona las operaciones relacionadas con los clientes.
- **GestionPlanes:** Gestiona las operaciones relacionadas con los planes.

Model: Define las entidades del dominio y la conexión con la base de datos.

- **Cliente:** Representa un cliente con sus atributos y métodos.
- **Factura:** Representa una factura generada para un cliente.
- **Plan:** Clase abstracta para los planes, con clases concretas como PlanPostPagoMegas, PlanPostPagoMinutos, PlanPostPagoMinutosMegas, y PlanPostPagoMinutosMegasEconomico.
- **PlanFactory:** Fabrica instancias de diferentes tipos de planes.
- **ConexionDB:** Maneja la conexión y las operaciones con la base de datos.

View: Proporciona la interfaz gráfica para la interacción del usuario.

- **FormClientes:** Formulario para gestionar clientes.
- **FormPlanes:** Formulario para gestionar planes.
- **Ejecutor:** Clase principal que inicia la aplicación.

Principales Funcionalidades

Gestión de Clientes

- **Agregar Cliente:** Permite agregar nuevos clientes tanto a la base de datos como a la lista local.

- **Actualizar Cliente:** Actualiza la información de un cliente existente en la base de datos y en la lista local.
- **Eliminar Cliente:** Elimina un cliente de la base de datos y de la lista local.
- **Asignar Plan a Cliente:** Asigna uno o dos planes a un cliente específico.
- **Generar Factura:** Calcula el costo total de los planes asignados a un cliente y genera una factura.
- **Mostrar Clientes:** Muestra la lista de clientes en una tabla.
- **Mostrar Facturas:** Muestra la lista de facturas generadas en una nueva ventana.

Gestión de Planes

- **Agregar Plan:** Permite agregar nuevos planes a la base de datos y a la lista local.
- **Actualizar Plan:** Actualiza la información de un plan existente.
- **Eliminar Plan:** Elimina un plan de la base de datos y de la lista local.
- **Mostrar Planes:** Muestra la lista de planes en una tabla.

Análisis del Código

- **Herencia y Polimorfismo**
  La implementación de diferentes tipos de planes utilizando una clase abstracta (Plan) y clases concretas (PlanPostPagoMegas, PlanPostPagoMinutos, PlanPostPagoMinutosMegas, PlanPostPagoMinutosMegasEconomico) es un ejemplo claro de herencia y polimorfismo. Esto permite extender fácilmente el sistema para admitir nuevos tipos de planes sin modificar el código existente.

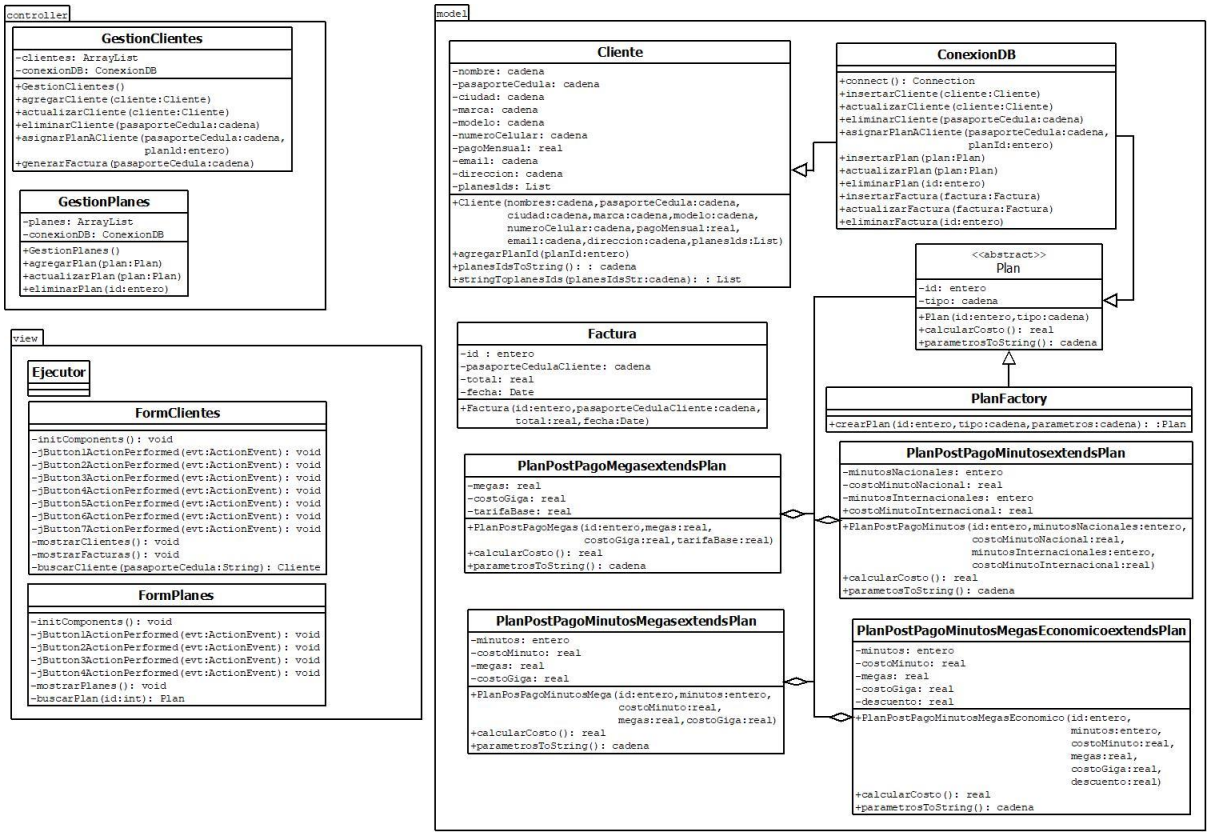
- **Patrón MVC**
  La separación de las responsabilidades en Modelo, Vista y Controlador facilita la mantenibilidad y escalabilidad del sistema. El controlador (GestionClientes y GestionPlanes) maneja la lógica del negocio y la comunicación con la base de datos, mientras que las vistas (FormClientes y FormPlanes) gestionan la interfaz de usuario.

- **Gestión de la Base de Datos**
  La clase ConexionDB encapsula todas las operaciones relacionadas con la base de datos, proporcionando métodos para insertar, actualizar, eliminar y obtener datos. Esto simplifica la interacción con la base de datos y permite cambiar fácilmente el motor de base de datos si es necesario.

- **Interacción con el Usuario**
  Las vistas utilizan JOptionPane para obtener entradas del usuario y mostrar mensajes, lo cual es adecuado para una aplicación de escritorio. Las tablas (JTable) proporcionan una forma clara y estructurada de mostrar datos a los usuarios.

En resumen, el programa de gestión de telefonía móvil estudiantil (Mov-UTPL) implementa de manera efectiva los principios de POO y el patrón MVC para gestionar clientes, planes y facturas. La estructura modular y la utilización de herencia y polimorfismo permiten una fácil extensibilidad y mantenibilidad del sistema. La separación de la lógica del negocio y la presentación asegura que el código sea limpio, organizado y fácil de entender. Este sistema es una sólida base para futuras mejoras y ampliaciones, y puede ser adaptado para otras aplicaciones de gestión similares.

Modelado UML



Fragmentos de código claves y su documentación:

A continuación, se presentan algunos fragmentos de código relevantes del programa, acompañados de su documentación para facilitar la comprensión de su funcionalidad:

1. Gestión de Clientes

```
package controller;

import model.Cliente;
import model.ConexionDB;
import model.Factura;
import model.Plan;

import java.util.ArrayList;
import java.util.Date;

/**
 * Clase que gestiona las operaciones relacionadas con los clientes.
 */
public class GestionClientes {
    private ArrayList<Cliente> clientes;
    private ConexionDB conexionDB;

    /**
     * Constructor que inicializa la conexión a la base de datos y obtiene la lista de clientes.
     */
    public GestionClientes() {
        this.conexionDB = new ConexionDB();
        this.clientes = conexionDB.obtenerClientes();
    }

    /**
     * Agrega un nuevo cliente a la base de datos y a la lista local.
     * @param cliente Cliente a agregar.
     */
    public void agregarCliente(Cliente cliente) {
        conexionDB.insertarCliente(cliente);
        clientes.add(cliente);
    }

    // Métodos adicionales para actualizar, eliminar y asignar planes a clientes
}
```

2. Gestión de Planes

```
3. package controller;
4.
5. import model.ConexionDB;
6. import model.Plan;
7. import java.util.ArrayList;
8.
9. /**
10.  * Clase que gestiona las operaciones relacionadas con los planes.
11.  */
12.public class GestionPlanes {
13.     private ArrayList<Plan> planes;
14.     private ConexionDB conexionDB;
15.
16.     /**
17.      * Constructor que inicializa la conexión a la base de datos y obtiene
   la lista de planes.
18.      */
19.     public GestionPlanes() {
20.         this.conexionDB = new ConexionDB();
21.         this.planes = conexionDB.obtenerPlanes();
22.     }
23.
24.     /**
25.      * Agrega un nuevo plan a la base de datos y a la lista local.
26.      * @param plan Plan a agregar.
27.      */
28.     public void agregarPlan(Plan plan) {
29.         conexionDB.insertarPlan(plan);
30.         planes.add(plan);
31.     }
32.
33.     // Métodos adicionales para actualizar y eliminar planes
34.}
35.
```

3. Gestión de Planes

```
4. package model;
5.
6. import java.util.ArrayList;
7. import java.util.Arrays;
8. import java.util.List;
9. import java.util.stream.Collectors;
10.
11./**
12. * Clase que representa un cliente.
13. */
14.
15.public class Cliente {
16.     private String nombres;
17.     private String pasaporteCedula;
18.     private String ciudad;
19.     private String marca;
20.     private String modelo;
21.     private String numeroCelular;
22.     private double pagoMensual;
23.     private String email;
24.     private String direccion;
25.     private List<Integer> planesIds;
26.
27.     public Cliente(String nombres, String pasaporteCedula, String ciudad,
    String marca, String modelo, String numeroCelular, double pagoMensual,
    String email, String direccion, List<Integer> planesIds) {
28.         this.nombres = nombres;
29.         this.pasaporteCedula = pasaporteCedula;
30.         this.ciudad = ciudad;
31.         this.marca = marca;
32.         this.modelo = modelo;
33.         this.numeroCelular = numeroCelular;
34.         this.pagoMensual = pagoMensual;
35.         this.email = email;
36.         this.direccion = direccion;
37.         this.planesIds = planesIds;
38.     }
39.
40.     // Métodos getter y setter
41.
42.     /**
43.      * Agrega un ID de plan a la lista del cliente, con un máximo de 2
    planes.
44.      * @param planId ID del plan a agregar.
45.      */
46.
```

```
47.     public void agregarPlanId(int planId) {
48.         if (this.planesIds.size() < 2) {
49.             this.planesIds.add(planId);
50.         } else {
51.             throw new IllegalStateException("El cliente no puede tener más
    de 2 planes.");
52.         }
53.     }
54.
55.     // Métodos adicionales para conversión de listas a cadenas y viceversa
56.}
```

4. Gestión de Planes

```
5. package model;
6.
7. import java.sql.*;
8. import java.util.ArrayList;
9.
10./**
11. * Clase que gestiona la conexión y operaciones con la base de datos.
12. */
13.public class ConexionDB {
14.     /**
15.      * Conecta a la base de datos SQLite.
16.      * @return Conexión a la base de datos.
17.      */
18.     private Connection connect() {
19.         String url = "jdbc:sqlite:./db/mov_utpl.db";
20.         Connection conn = null;
21.         try {
22.             conn = DriverManager.getConnection(url);
23.         } catch (SQLException e) {
24.             System.out.println(e.getMessage());
25.         }
26.         return conn;
27.     }
28.
29.     /**
30.      * Inserta un cliente en la base de datos.
31.      * @param cliente Cliente a insertar.
32.      */
33.     public void insertarCliente(Cliente cliente) {
34.         String sql = "INSERT INTO clientes(nombres, pasaporteCedula,
    ciudad, marca, modelo, numeroCelular, pagoMensual, email, direccion,
    planesIds) VALUES(?,?,?,?,?,?,?,?,?,?)";
35.
```

```
36.         try (Connection conn = this.connect();
37.              PreparedStatement pstmt = conn.prepareStatement(sql)) {
38.             pstmt.setString(1, cliente.getNombres());
39.             pstmt.setString(2, cliente.getPasaporteCedula());
40.             pstmt.setString(3, cliente.getCiudad());
41.             pstmt.setString(4, cliente.getMarca());
42.             pstmt.setString(5, cliente.getModelo());
43.             pstmt.setString(6, cliente.getNumeroCelular());
44.             pstmt.setDouble(7, cliente.getPagoMensual());
45.             pstmt.setString(8, cliente.getEmail());
46.             pstmt.setString(9, cliente.getDireccion());
47.             pstmt.setString(10, cliente.planesIdsToString());
48.             pstmt.executeUpdate();
49.         } catch (SQLException e) {
50.             System.out.println(e.getMessage());
51.         }
52.     }
53.
54.     // Métodos adicionales para obtener, actualizar y eliminar clientes,
    planes y facturas
55. }
56.
```

URL en GIT del programa:

https://github.com/ProOrientadaObjetos-P-D-AA2024/aab2-25-proyecto-bimestral-2do-bim-grupal-arnautdj.git

Captura del Funcionamiento: