#### 

**Universidad Técnica Particular de Loja**

Programación Orientada a Objetos

Proyecto Bimestral Grupal

Logotipo

Descripción generada automáticamente

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**

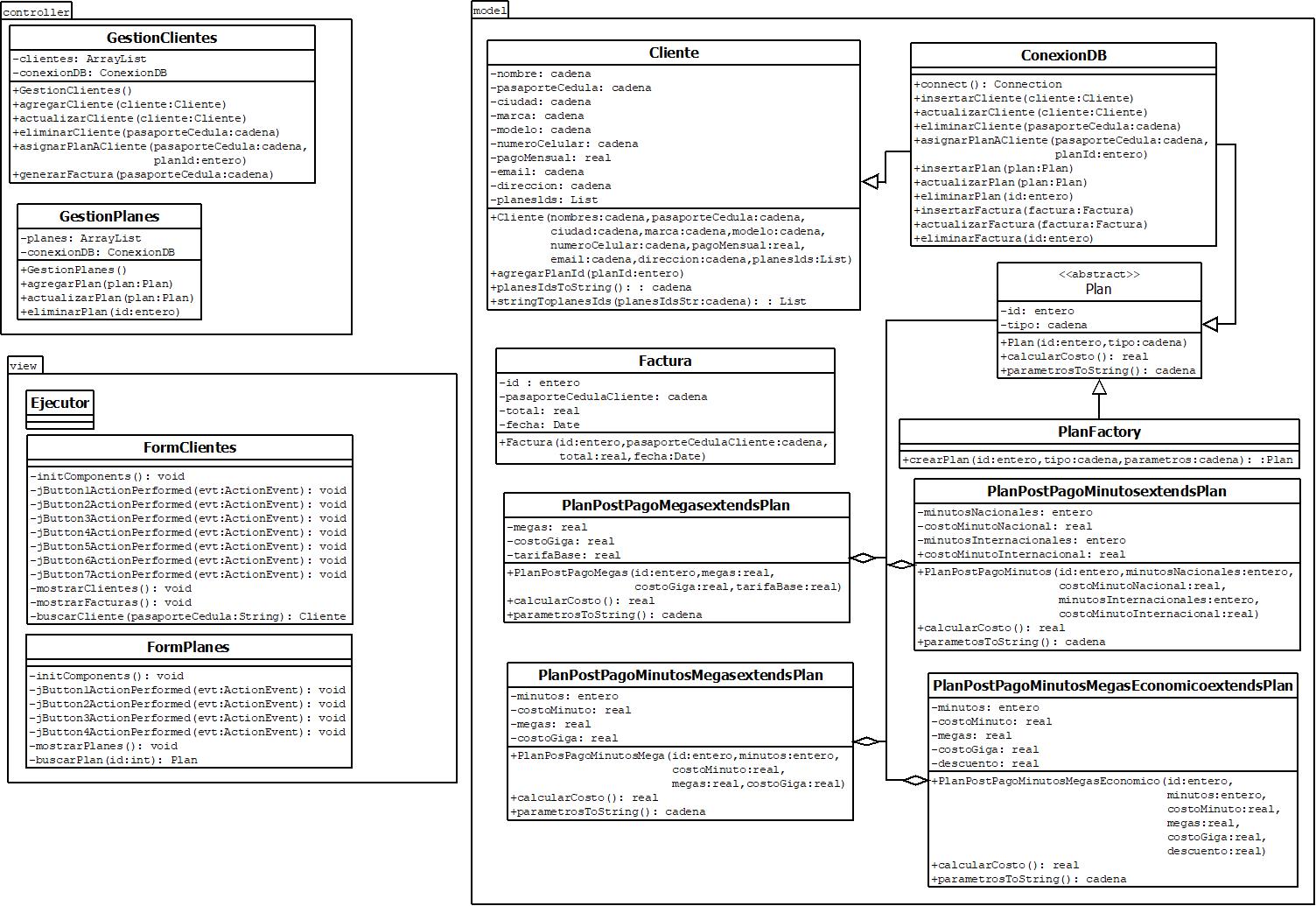
La clase ConexionDB encapsula todas las operaciones relacionadas con la base de datos, proporcionando métodos para insertar, actualizar, eliminar y obtener datos. Esto simplifica la interacción con la base de datos y permite cambiar fácilmente el motor de base de datos si es necesario.

* **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**
2. package controller;
3. import model.Cliente;
4. import model.ConexionDB;
5. import model.Factura;
6. import model.Plan;
7. import java.util.ArrayList;
8. import java.util.Date;
9. /\*\*
10. \* Clase que gestiona las operaciones relacionadas con los clientes.
11. \*/
12. public class GestionClientes {
13. private ArrayList<Cliente> clientes;
14. private ConexionDB conexionDB;
15. /\*\*
16. \* Constructor que inicializa la conexión a la base de datos y obtiene la lista de clientes.
17. \*/
18. public GestionClientes() {
19. this.conexionDB = new ConexionDB();
20. this.clientes = conexionDB.obtenerClientes();
21. }
22. /\*\*
23. \* Agrega un nuevo cliente a la base de datos y a la lista local.
24. \* @param cliente Cliente a agregar.
25. \*/
26. public void agregarCliente(Cliente cliente) {
27. conexionDB.insertarCliente(cliente);
28. clientes.add(cliente);
29. }
30. // Métodos adicionales para actualizar, eliminar y asignar planes a clientes
31. }
32. **Gestión de Planes**
33. package controller;
34. import model.ConexionDB;
35. import model.Plan;
36. import java.util.ArrayList;
37. /\*\*
38. \* Clase que gestiona las operaciones relacionadas con los planes.
39. \*/
40. public class GestionPlanes {
41. private ArrayList<Plan> planes;
42. private ConexionDB conexionDB;
43. /\*\*
44. \* Constructor que inicializa la conexión a la base de datos y obtiene la lista de planes.
45. \*/
46. public GestionPlanes() {
47. this.conexionDB = new ConexionDB();
48. this.planes = conexionDB.obtenerPlanes();
49. }
50. /\*\*
51. \* Agrega un nuevo plan a la base de datos y a la lista local.
52. \* @param plan Plan a agregar.
53. \*/
54. public void agregarPlan(Plan plan) {
55. conexionDB.insertarPlan(plan);
56. planes.add(plan);
57. }
58. // Métodos adicionales para actualizar y eliminar planes
59. }
60. **Gestión de Planes**
61. package model;
62. import java.util.ArrayList;
63. import java.util.Arrays;
64. import java.util.List;
65. import java.util.stream.Collectors;
66. /\*\*
67. \* Clase que representa un cliente.
68. \*/
69. public class Cliente {
70. private String nombres;
71. private String pasaporteCedula;
72. private String ciudad;
73. private String marca;
74. private String modelo;
75. private String numeroCelular;
76. private double pagoMensual;
77. private String email;
78. private String direccion;
79. private List<Integer> planesIds;
80. public Cliente(String nombres, String pasaporteCedula, String ciudad, String marca, String modelo, String numeroCelular, double pagoMensual, String email, String direccion, List<Integer> planesIds) {
81. this.nombres = nombres;
82. this.pasaporteCedula = pasaporteCedula;
83. this.ciudad = ciudad;
84. this.marca = marca;
85. this.modelo = modelo;
86. this.numeroCelular = numeroCelular;
87. this.pagoMensual = pagoMensual;
88. this.email = email;
89. this.direccion = direccion;
90. this.planesIds = planesIds;
91. }
92. // Métodos getter y setter
93. /\*\*
94. \* Agrega un ID de plan a la lista del cliente, con un máximo de 2 planes.
95. \* @param planId ID del plan a agregar.
96. \*/
97. public void agregarPlanId(int planId) {
98. if (this.planesIds.size() < 2) {
99. this.planesIds.add(planId);
100. } else {
101. throw new IllegalStateException("El cliente no puede tener más de 2 planes.");
102. }
103. }
104. // Métodos adicionales para conversión de listas a cadenas y viceversa
105. }
106. **Gestión de Planes**
107. package model;
108. import java.sql.\*;
109. import java.util.ArrayList;
110. /\*\*
111. \* Clase que gestiona la conexión y operaciones con la base de datos.
112. \*/
113. public class ConexionDB {
114. /\*\*
115. \* Conecta a la base de datos SQLite.
116. \* @return Conexión a la base de datos.
117. \*/
118. private Connection connect() {
119. String url = "jdbc:sqlite:./db/mov\_utpl.db";
120. Connection conn = null;
121. try {
122. conn = DriverManager.getConnection(url);
123. } catch (SQLException e) {
124. System.out.println(e.getMessage());
125. }
126. return conn;
127. }
128. /\*\*
129. \* Inserta un cliente en la base de datos.
130. \* @param cliente Cliente a insertar.
131. \*/
132. public void insertarCliente(Cliente cliente) {
133. String sql = "INSERT INTO clientes(nombres, pasaporteCedula, ciudad, marca, modelo, numeroCelular, pagoMensual, email, direccion, planesIds) VALUES(?,?,?,?,?,?,?,?,?,?)";
134. try (Connection conn = this.connect();
135. PreparedStatement pstmt = conn.prepareStatement(sql)) {
136. pstmt.setString(1, cliente.getNombres());
137. pstmt.setString(2, cliente.getPasaporteCedula());
138. pstmt.setString(3, cliente.getCiudad());
139. pstmt.setString(4, cliente.getMarca());
140. pstmt.setString(5, cliente.getModelo());
141. pstmt.setString(6, cliente.getNumeroCelular());
142. pstmt.setDouble(7, cliente.getPagoMensual());
143. pstmt.setString(8, cliente.getEmail());
144. pstmt.setString(9, cliente.getDireccion());
145. pstmt.setString(10, cliente.planesIdsToString());
146. pstmt.executeUpdate();
147. } catch (SQLException e) {
148. System.out.println(e.getMessage());
149. }
150. }
151. // Métodos adicionales para obtener, actualizar y eliminar clientes, planes y facturas
152. }

**URL en GIT del programa:**

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

**Captura del Funcionamiento:**

