UTPL
La Universidad Católica de Loja

MOV-UTPL

GESTIÓN DE TELEFONÍA MÓVIL ESTUDIANTIL

Programación Orientada a Objetos

Paralelo D

Marco S. Herrera M.
msherrera13@utpl.edu.ec

Carlos S. Valladares C.
csvalladares2@utpl.edu.ec

1. Análisis

1.1. Objetivo

Desarrollar un sistema de gestión de clientes y facturación para un servicio de comunicación móvil dirigido a la comunidad universitaria de la Universidad Técnica Particular de Loja (UTPL). El sistema debe permitir la administración de clientes y planes móviles, así como la generación de facturas, utilizando una base de datos SQLite y aplicando la arquitectura MVC.

1.2. Requisitos

1.2.1. Gestión de Clientes

- Registro de nuevos clientes con datos básicos: nombres, pasaporte, ciudad, marca del dispositivo, número de celular, número de tarjeta de crédito, contraseña y pago mensual.
- Actualización, eliminación y consulta de información de clientes.
- Restricción a un máximo de dos tipos de planes móviles por cliente.

1.2.2. Gestión de Planes Móviles

Creación y administración de los siguientes tipos de planes móviles:

- **PlanPostPagoMinutosMegasEconomico**: Información sobre minutos, costo por minuto, megas en gigas, costo por giga, y porcentaje de descuento.
- **PlanPostPagoMinutos**: Información sobre minutos nacionales, costo por minuto nacional, minutos internacionales, y costo por minuto internacional.
- **PlanPostPagoMegas**: Información sobre megas en gigas, costo por giga, y tarifa base.
- **PlanPostPagoMinutosMegas**: Información sobre minutos, costo por minuto, megas en gigas, y costo por giga.

1.2.3. Generación de Facturas

- Cálculo y emisión de facturas basadas en los planes contratados por cada cliente, reflejando los costos correspondientes.

1.2.4. Base de Datos

- Implementación del motor de base de datos SQLite para la gestión de clientes, planes y facturas.
- Diseño de tablas para clientes, planes móviles y facturas, con relaciones adecuadas.

1.2.5. Interfaz de Usuario

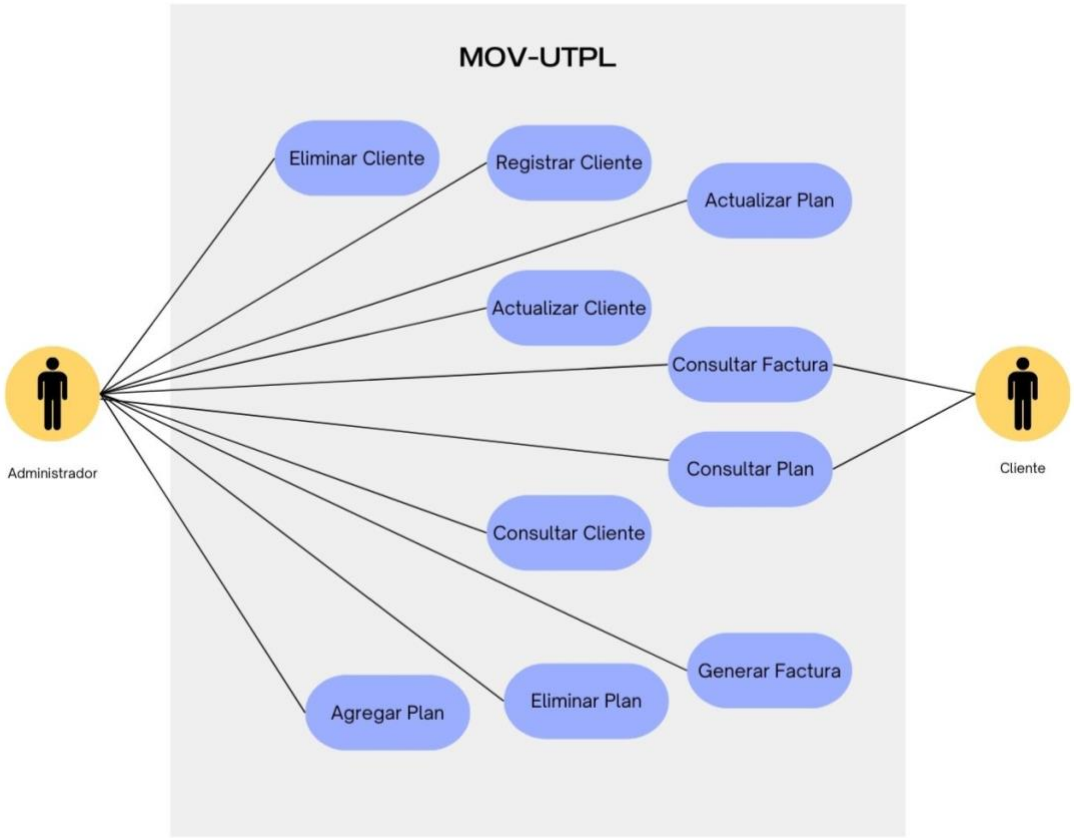
- Desarrollo de interfaces para realizar operaciones CRUD (Crear, Leer, Actualizar, Eliminar) en la gestión de clientes y planes.

1.3. Tabla de Requisitos Específicos

Clase	Atributos	Operaciones
Cientes	Nombre, pasaporte, ciudad, marca del dispositivo, número de celular, número de tarjeta de crédito, contraseña y pago mensual.	Crear, leer, actualizar y eliminar clientes. Validación de datos de entrada y autenticación de usuarios.
Planes Móviles	Cada tipo de plan debe almacenar información específica como minutos, megas, costos y descuentos.	Crear, leer, actualizar y eliminar planes. Aplicación de cálculos de costos y descuentos según el plan.

1.4. Diagrama de Casos de Uso

1.4.1. Diagrama



1.4.2. Actores

- **Administrador**: Persona encargada de gestionar clientes, planes y facturación.
- **Cliente**: Usuario del sistema que tiene planes móviles y recibe facturación.

1.4.3. Casos de Uso

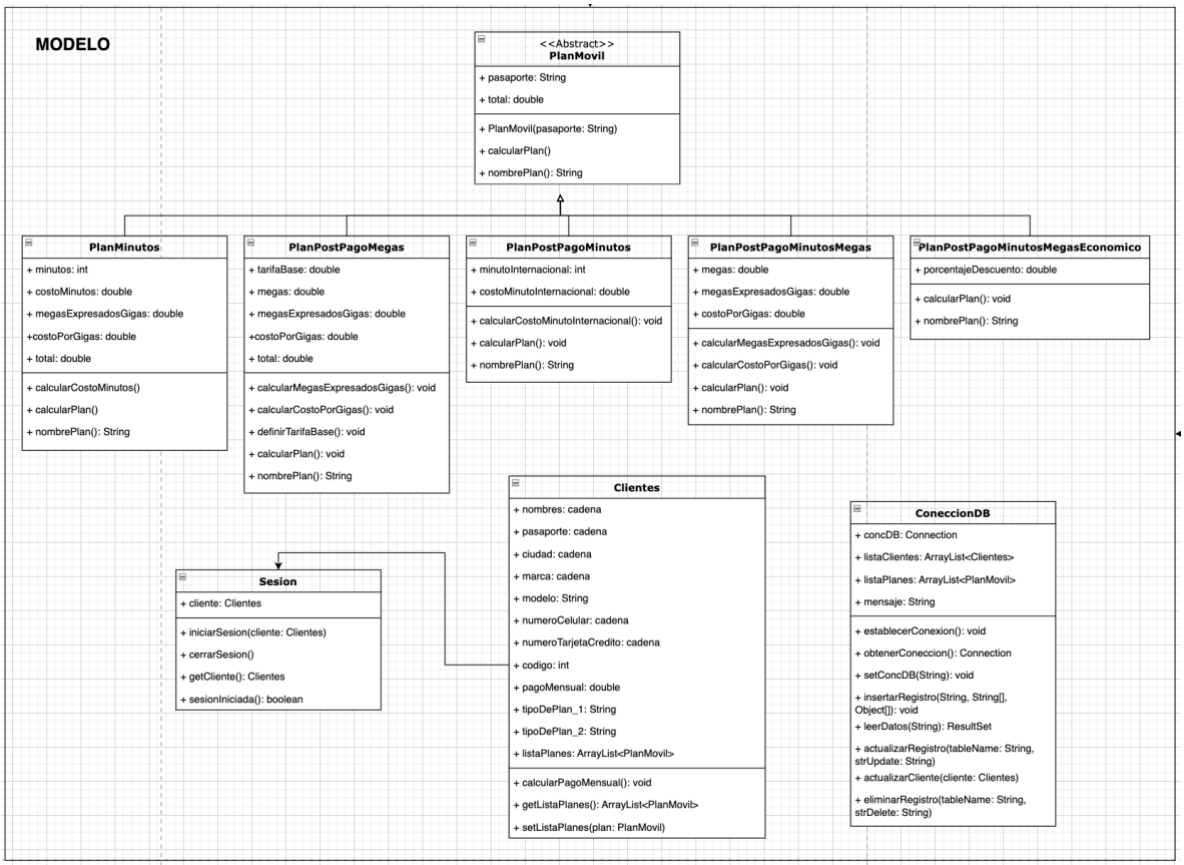
- **Registrar Cliente**: Permite al Administrador agregar un nuevo cliente al sistema.
- **Actualizar Cliente**: Permite al Administrador modificar la información de un cliente existente.
- **Eliminar Cliente**: Permite al Administrador eliminar un cliente del sistema.
- **Consultar Cliente**: Permite al Administrador buscar y visualizar la información de un cliente.
- **Agregar Plan**: Permite al Administrador crear un nuevo tipo de plan móvil.
- **Actualizar Plan**: Permite al Administrador modificar los detalles de un plan existente.
- **Eliminar Plan**: Permite al Administrador eliminar un plan móvil del sistema.
- **Consultar Plan**: Permite al Administrador y al Cliente buscar y visualizar la información de un plan.
- **Generar Factura**: Permite al Administrador generar y emitir una factura para un cliente basado en los planes contratados.
- **Consultar Factura**: Permite al Administrador y al Cliente consultar el historial de facturas de un cliente.

2. Diseño

Arquitectura y componentes del sistema de gestión de clientes y facturación, asegurando que cumpla con los requisitos establecidos y se pueda implementar de manera eficiente utilizando la arquitectura MVC y una base de datos SQLite.

2.1. Arquitectura del Sistema: Modelo-Vista-Controlador (MVC)

2.1.1. Modelo



2.1.2.1. Clientes

La clase Clientes representa a un cliente del sistema. Contiene atributos básicos sobre el cliente y métodos para manejar los planes móviles asociados a él.

Atributos:

- **String nombres, pasaporte, ciudad, marca, modelo, numeroCelular, numeroTarjetaCredito, tipoDePlan_1, tipoDePlan_2**: Información personal y de contacto del cliente.

- **int codigo**: Identificador único del cliente.
- **double pagoMensual**: Pago mensual que el cliente debe realizar.
- **ArrayList<PlanMovil> listaPlanes**: Lista de planes móviles asociados al cliente.

Métodos:

- **Constructores**: Para crear instancias de Clientes con diferentes combinaciones de atributos.
- **Getters y Setters**: Para acceder y modificar los atributos del cliente.
- **calcularPagoMensual()**: Calcula el total del pago mensual sumando el costo de los planes móviles.
- **setListaPlanes(PlanMovil plan)**: Añade un nuevo plan a la lista, asegurando que el cliente no tenga más de dos planes.
- **toString()**: Representación en cadena de los atributos del cliente y sus planes.

2.1.2.2. ConeccionDB

La clase ConeccionDB maneja la conexión con la base de datos y proporciona métodos para interactuar con ella.

Atributos:

- **Connection concDB**: Objeto para la conexión con la base de datos.
- **ArrayList<Clientes> listaClientes**: Lista de clientes (no se usa en el código proporcionado).
- **ArrayList<PlanMovil> listaPlanes**: Lista de planes (no se usa en el código proporcionado).
- **String mensaje**: Mensaje para errores en operaciones con la base de datos.

Métodos:

- **establecerConexion()**: Establece la conexión con la base de datos.
- **obtenerConeccion()**: Devuelve la conexión actual.
- **setConcDB(String url)**: Establece la conexión con la base de datos usando una URL dada.
- **insertarRegistro(), leerDatos(), obtenerCodigoPorPasaporte(), obtenerClientePorPasaporte(), actualizarRegistro(), actualizarCliente(), eliminarRegistro()**: Métodos CRUD para interactuar con la base de datos.

2.1.2.3. PlanMovil (Clase abstracta)

PlanMovil es una clase abstracta que define la interfaz para los diferentes tipos de planes móviles.

Atributos:

- **String pasaporte**: Identificador del cliente asociado al plan.
- **double total**: Costo total del plan.

Métodos:

- **constructores**: Inicializan el pasaporte.
- **abstract void calcularPlan()**: Método para calcular el costo del plan (implementado en clases derivadas).
- **abstract String nombrePlan()**: Devuelve el nombre del plan (implementado en clases derivadas).
- **Getters y Setters**: Para el pasaporte y el total.

2.1.2.4. PlanMinutos, PlanPostPagoMegas, PlanPostPagoMinutos, PlanPostPago MinutosMegas, PlanPostPagoMinutosMegasEconomico

Estas clases representan distintos tipos de planes móviles y extienden PlanMovil.

Métodos Específicos:

- **PlanMinutos**: Calcula el costo de los minutos. No implementa los métodos abstractos completamente.
- **PlanPostPagoMegas**: Maneja el costo basado en megas, incluyendo tarifas base y costo por gigas.
- **PlanPostPagoMinutos**: Maneja el costo de minutos y minutos internacionales.
- **PlanPostPagoMinutosMegas**: Maneja tanto minutos como megas.
- **PlanPostPagoMinutosMegasEconomico**: Similar a PlanPostPagoMinutos Megas, pero con un descuento aplicado.

2.1.2.4. Sesion

La clase Sesion maneja la sesión de un cliente en el sistema.

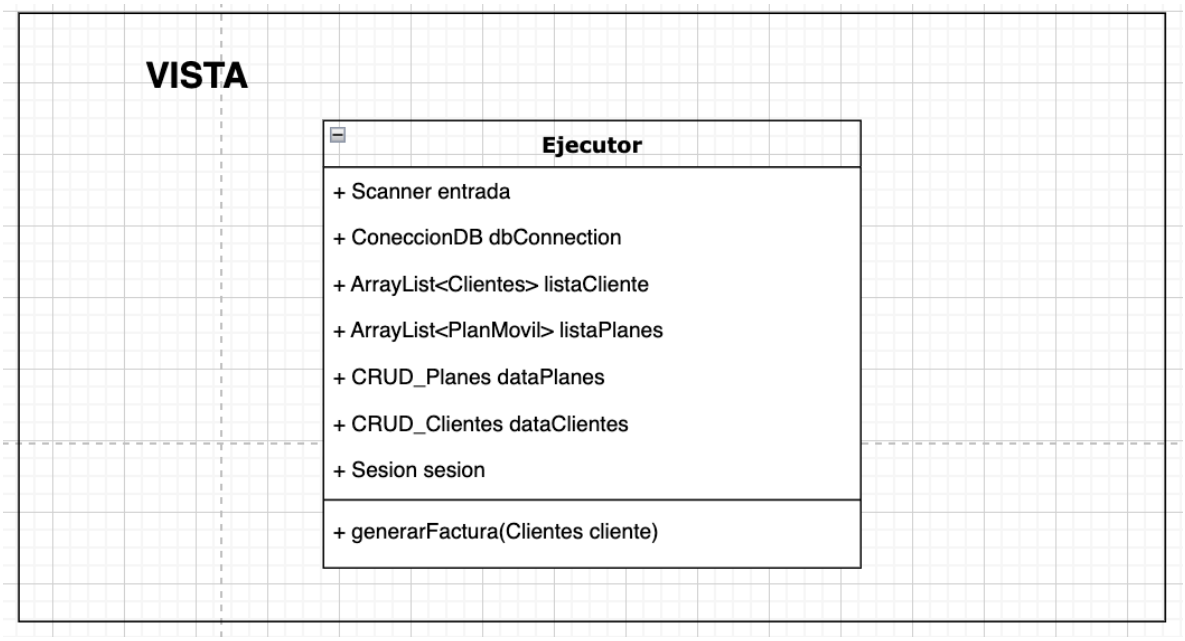
Atributos:

- **Clientes cliente**: Cliente actualmente en sesión.

Métodos:

- **iniciarSesion(Clientes cliente)**: Inicia una sesión para el cliente.
- **cerrarSesion()**: Cierra la sesión del cliente.
- **getCliente()**: Devuelve el cliente en sesión.
- **sesionIniciada()**: Verifica si hay una sesión activa.

2.1.2. Vista



1. **Interfaz de Usuario (UI):**

La Vista en tu código se encarga de la interacción con el usuario a través de la consola. Muestra menús y mensajes, y recoge la entrada del usuario.

Se utilizan comandos y opciones para realizar diversas acciones como iniciar sesión, registrarse, comprar o eliminar planes, y generar facturas.

2. **Presentación de Datos:**

Muestra datos al usuario en forma de texto. Por ejemplo, muestra mensajes sobre el estado de las operaciones (éxito o error) y detalles de los planes y clientes.

Presenta la factura del cliente con información detallada sobre el cliente y los planes comprados.

3. **Interacción con el Controlador:**

La Vista realiza operaciones basadas en la selección del usuario y luego invoca métodos de los controladores (CRUD_Planes, CRUD_Clientes) para manipular los datos.

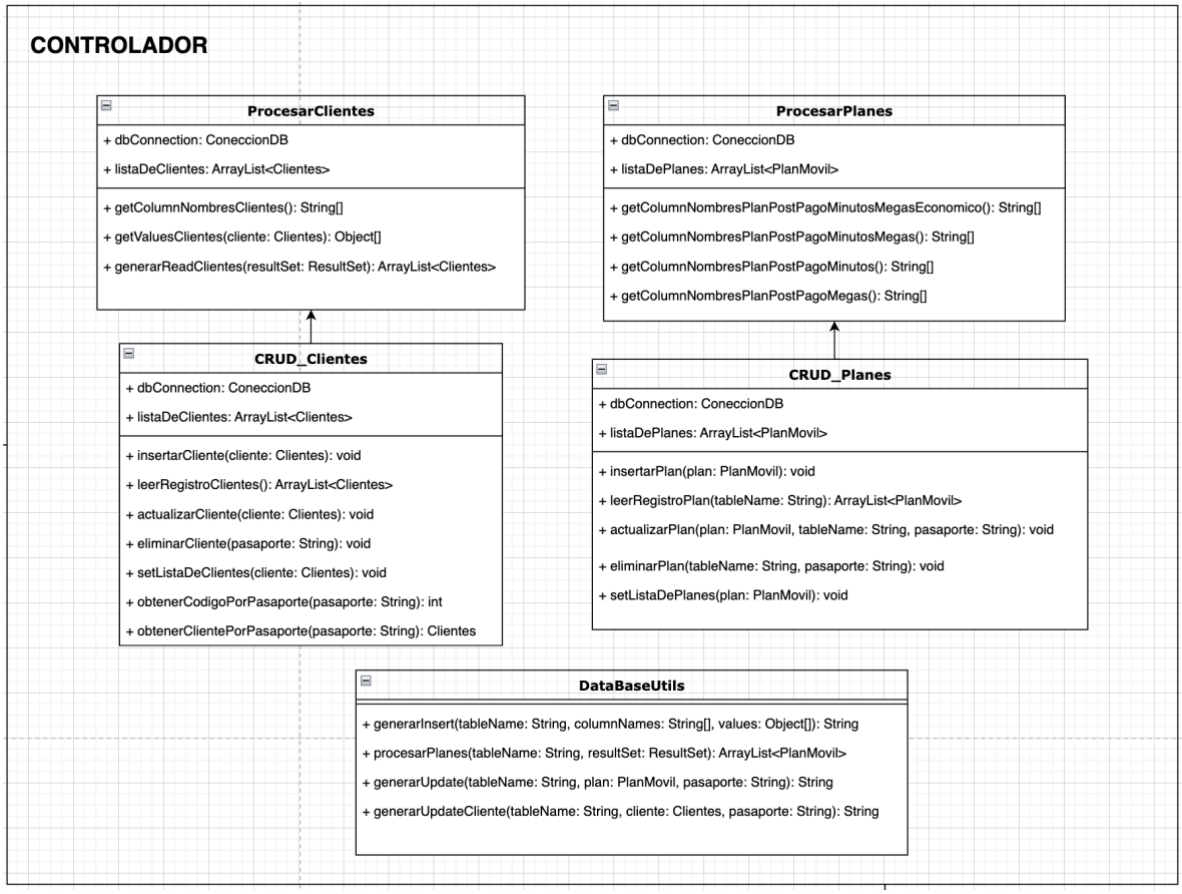
4. **Recogida de Datos del Usuario:**

Solicita y recoge la entrada del usuario mediante la clase Scanner. Los datos ingresados son luego utilizados para realizar operaciones como la compra de planes o el registro de nuevos clientes.

5. **Validación de Entrada:**

Aunque la Vista realiza algunas validaciones básicas (como verificar que el pasaporte y la contraseña coincidan), la validación más profunda y las reglas de negocio se manejan en el Modelo o el Controlador.

2.1.3. Controlador



2.1.3.1. CRUD_Clientes

- Maneja operaciones CRUD (Crear, Leer, Actualizar, Eliminar) para la entidad Clientes.
- Interactúa con la base de datos a través de la clase ConeccionDB.

Métodos:

- insertarCliente(Clientes cliente): Inserta un nuevo cliente en la base de datos.
- leerRegistroClientes(): Lee todos los registros de clientes desde la base de datos.
- actualizarCliente(Clientes cliente): Actualiza la información de un cliente específico.
- eliminarCliente(String pasaporte): Elimina un cliente basado en el pasaporte.
- setListaDeClientes(Clientes cliente): Añade un cliente a la lista interna de clientes.
- obtenerCodigoPorPasaporte(String pasaporte): Obtiene el código de cliente usando el pasaporte.
- obtenerClientePorPasaporte(String pasaporte): Obtiene un cliente específico usandoel pasaporte.

2.1.3.2. CRUD_Planes

- Maneja operaciones CRUD para la entidad PlanMovil y sus subtipos (PlanPostPagoMinutosMegasEconomico, PlanPostPagoMinutosMegas, PlanPostPagoMinutos, PlanPostPagoMegas).
- Interactúa con la base de datos para insertar, leer, actualizar y eliminar planes.

Métodos Clave

- insertarPlan(PlanMovil plan): Inserta un nuevo plan en la base de datos según el tipo de plan.
- leerRegistroPlan(String tableName): Lee todos los registros de un tipo específico de plan.
- actualizarPlan(PlanMovil plan, String tableName, String pasaporte): Actualiza un plan específico en la base de datos.
- eliminarPlan(String tableName, String pasaporte): Elimina un plan basado en el pasaporte.
- setListaDePlanes(PlanMovil plan): Añade un plan a la lista interna de planes.

2.1.3.3. DataBaseUtils

- Proporciona utilidades para generar consultas SQL para inserciones y actualizaciones.

- **Métodos Clave**:

  - generarInsert(String tableName, String[] columnNames, Object[] values): Genera una consulta SQL para insertar datos.
  - generarUpdate(String tableName, PlanMovil plan, String pasaporte): Genera una consulta SQL para actualizar datos basados en el tipo de plan.
  - generarUpdateCliente(String tableName, Clientes cliente, String pasaporte): Genera una consulta SQL para actualizar datos del cliente.

2.1.3.4. ProcesarClientes

- Maneja la preparación y procesamiento de datos de clientes.

- **Métodos**:

  - getColumnNombresClientes(): Devuelve los nombres de columnas para la tabla de clientes.
  - getValuesClientes(Clientes cliente): Obtiene los valores de un cliente para una operación de inserción o actualización.
  - generarReadClientes(ResultSet resultSet): Genera una lista de objetos Clientes a partir de un ResultSet.

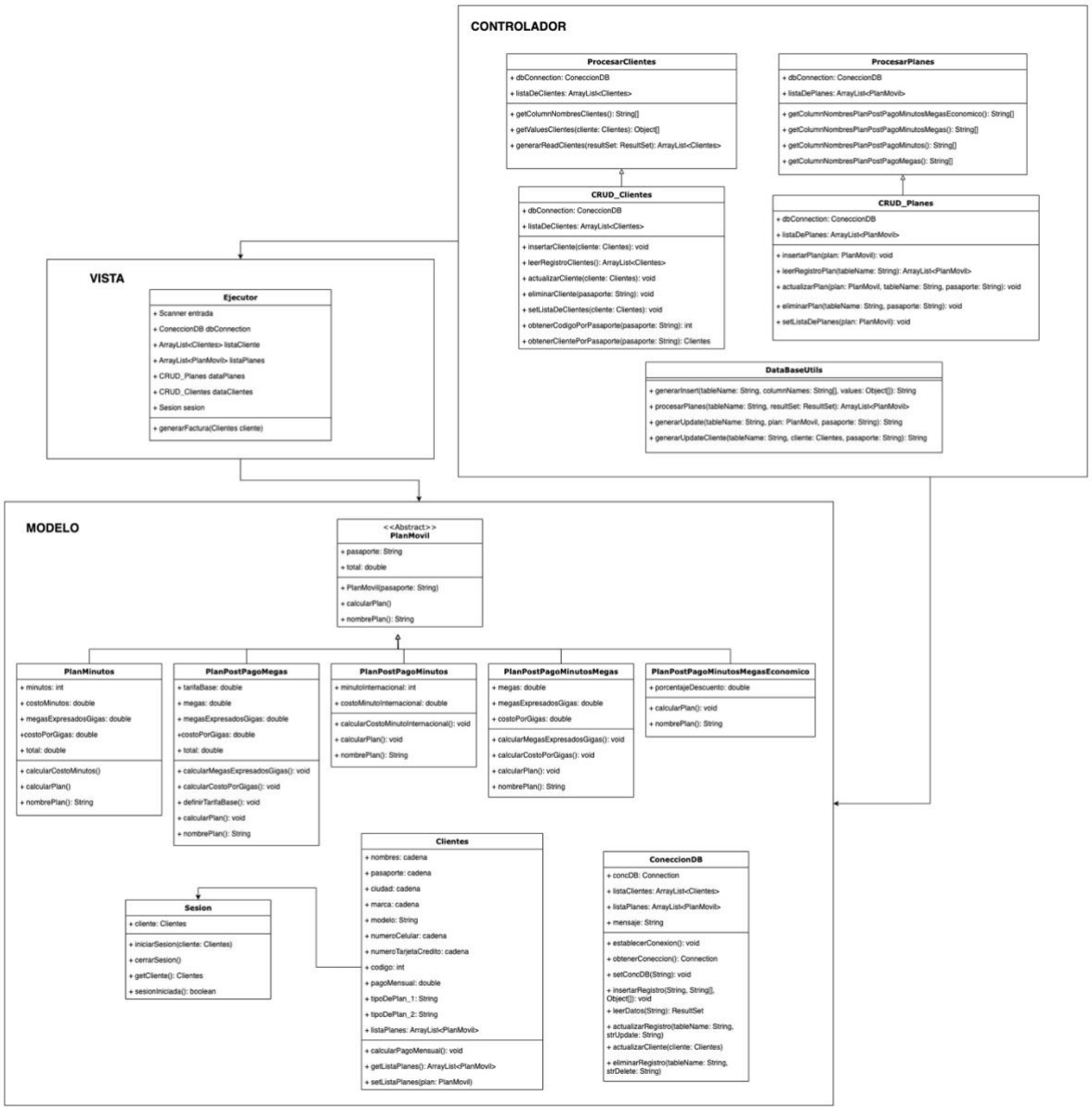
2.1.3.5. ProcesarPlanes

Maneja la preparación y procesamiento de datos de planes.

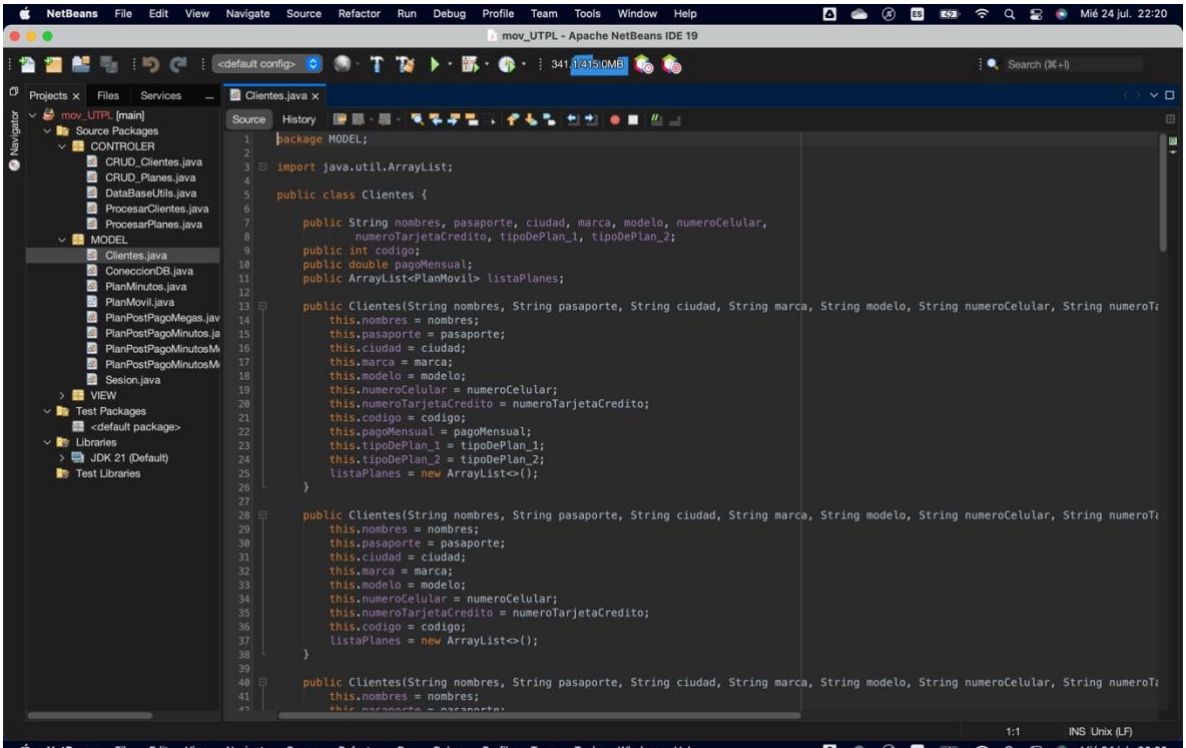
- **Métodos**:

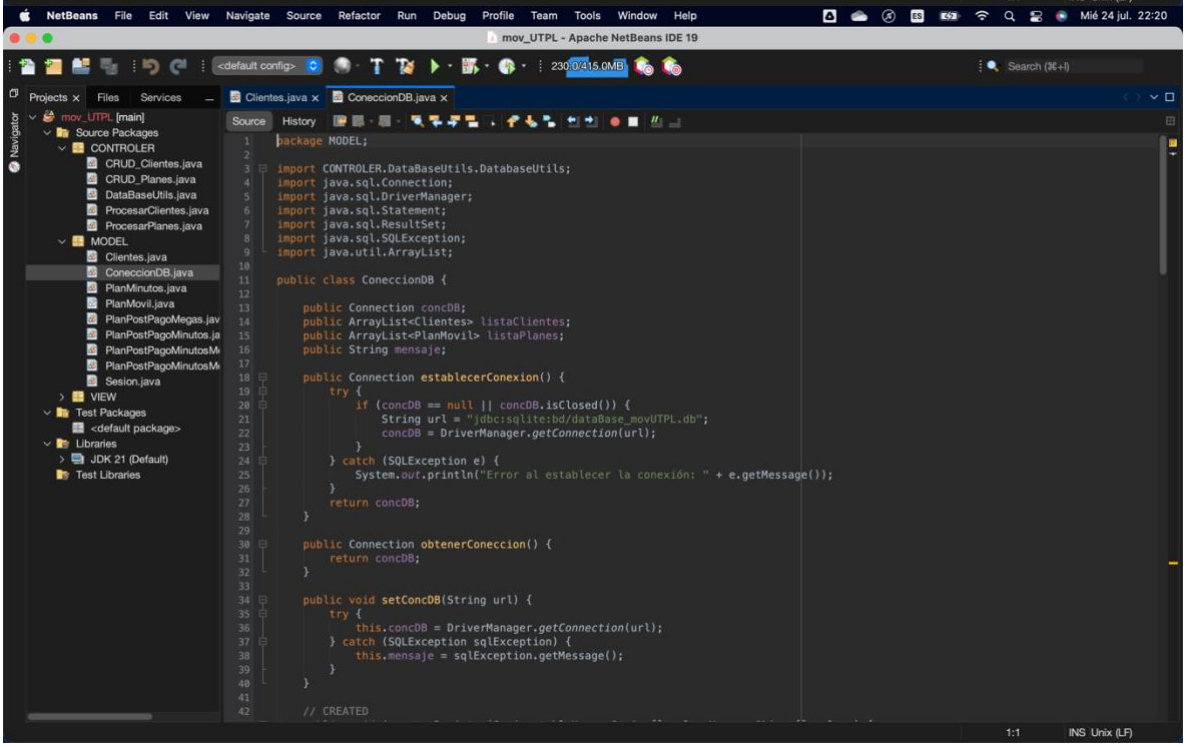
  - getColumnNombresPlanPostPagoMinutosMegasEconomico(): Devuelve los nombres de columnas para la tabla de PlanPostPagoMinutosMegasEconomico.
  - getValuesPlanPostPagoMinutosMegasEconomico(PlanPostPagoMinutosMegas Economico plan): Obtiene los valores de un plan específico.
  - procesarPlanes(String tableName, ResultSet resultSet): Genera una lista de objetos PlanMovil a partir de un ResultSet.

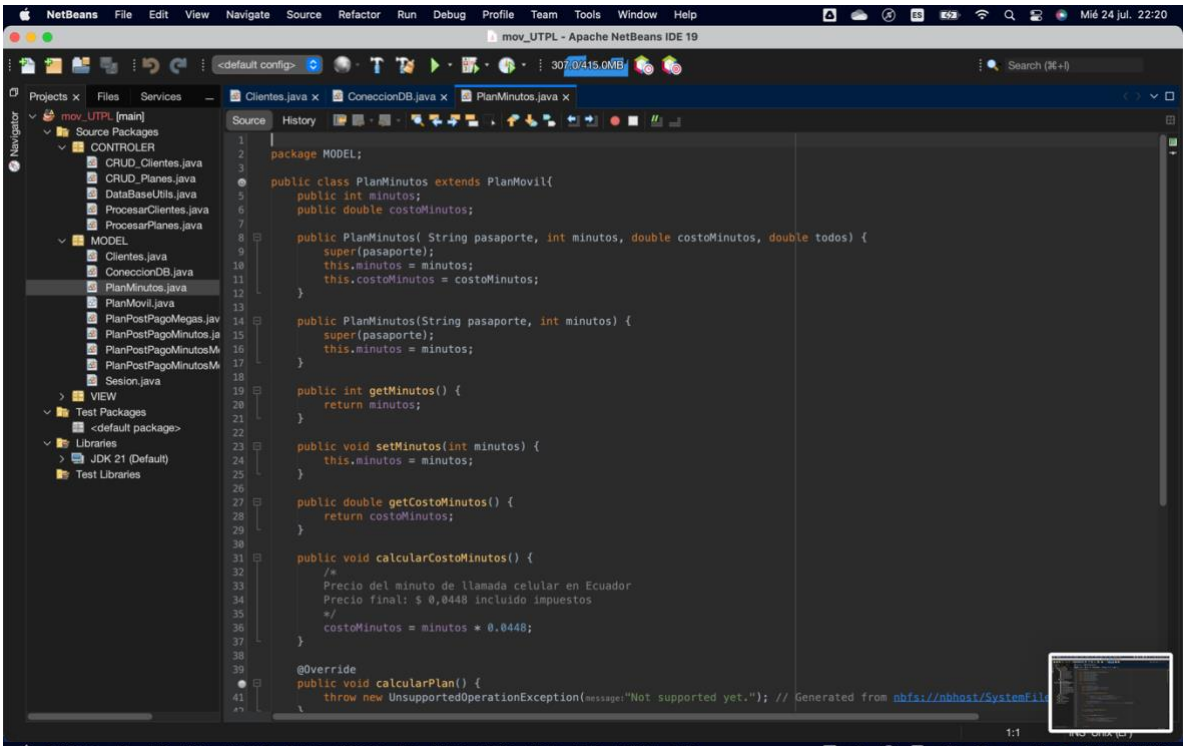
2.2. Diagrama UML Completo

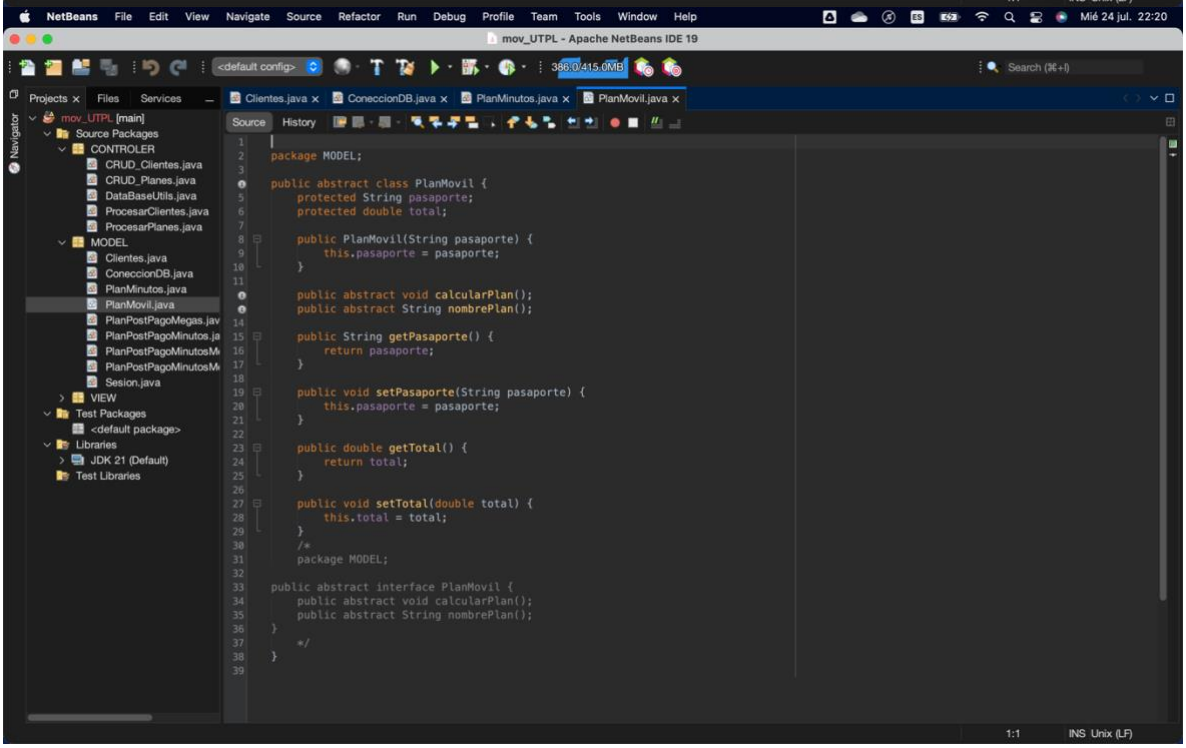


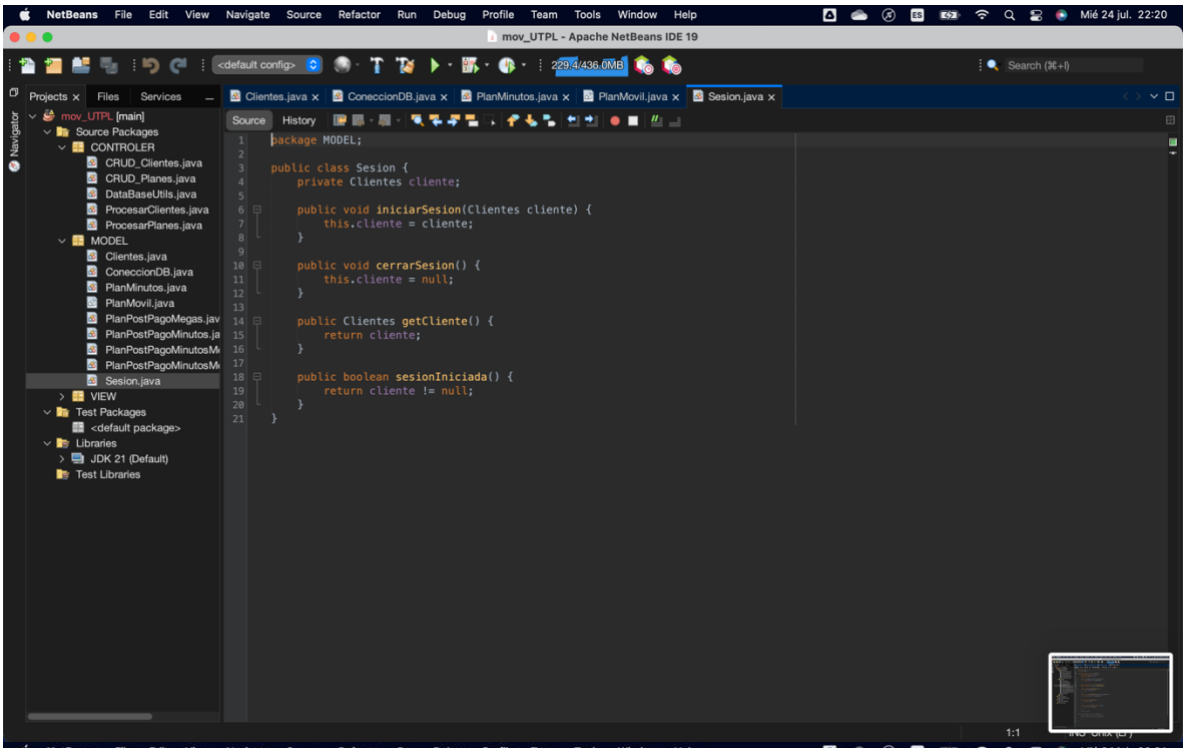
3. Codificación

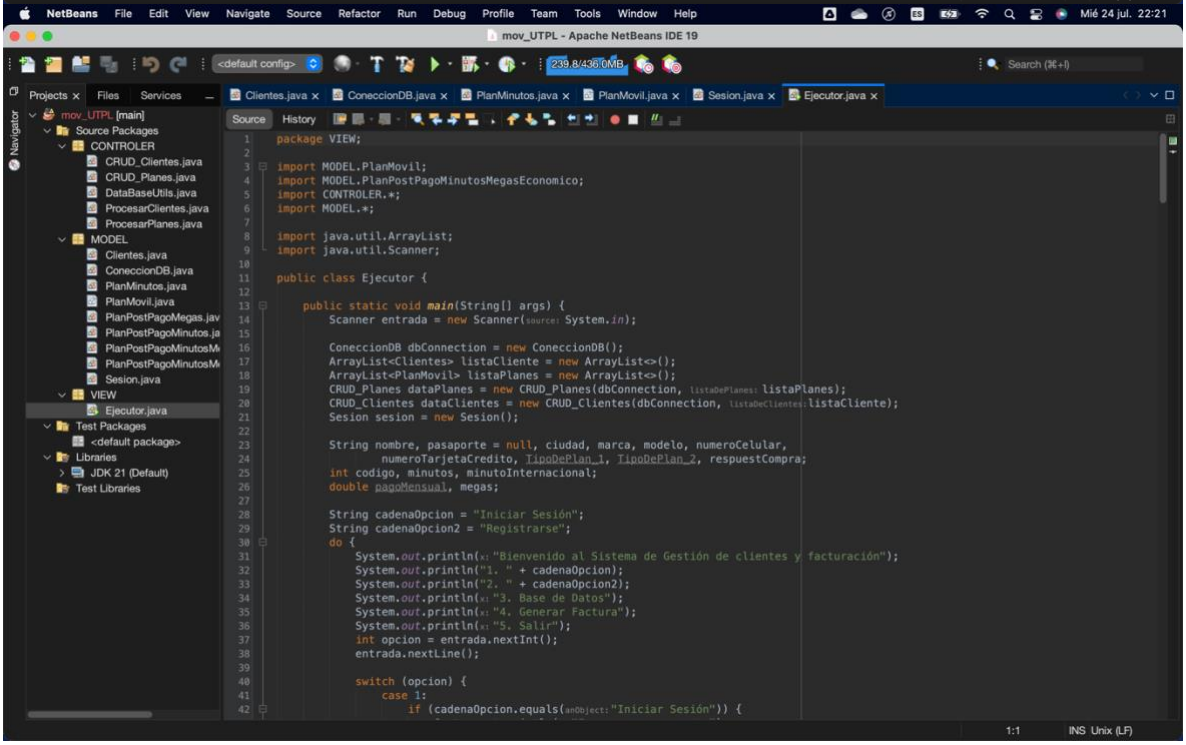


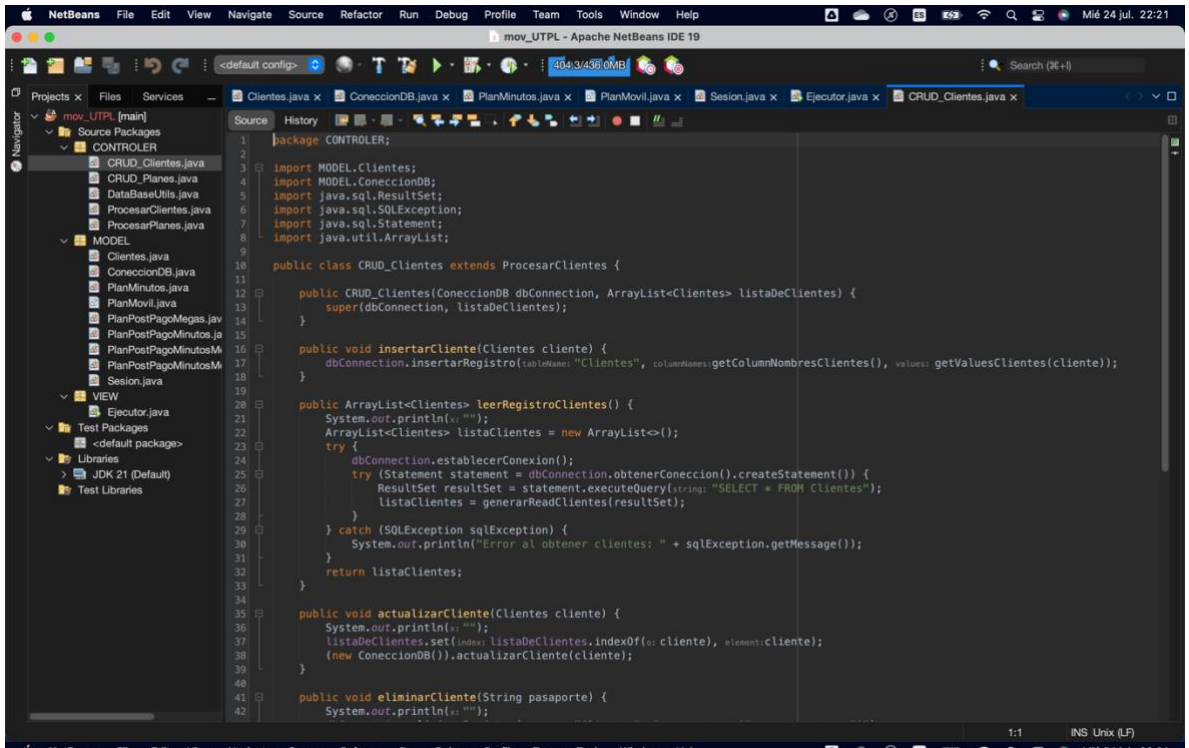


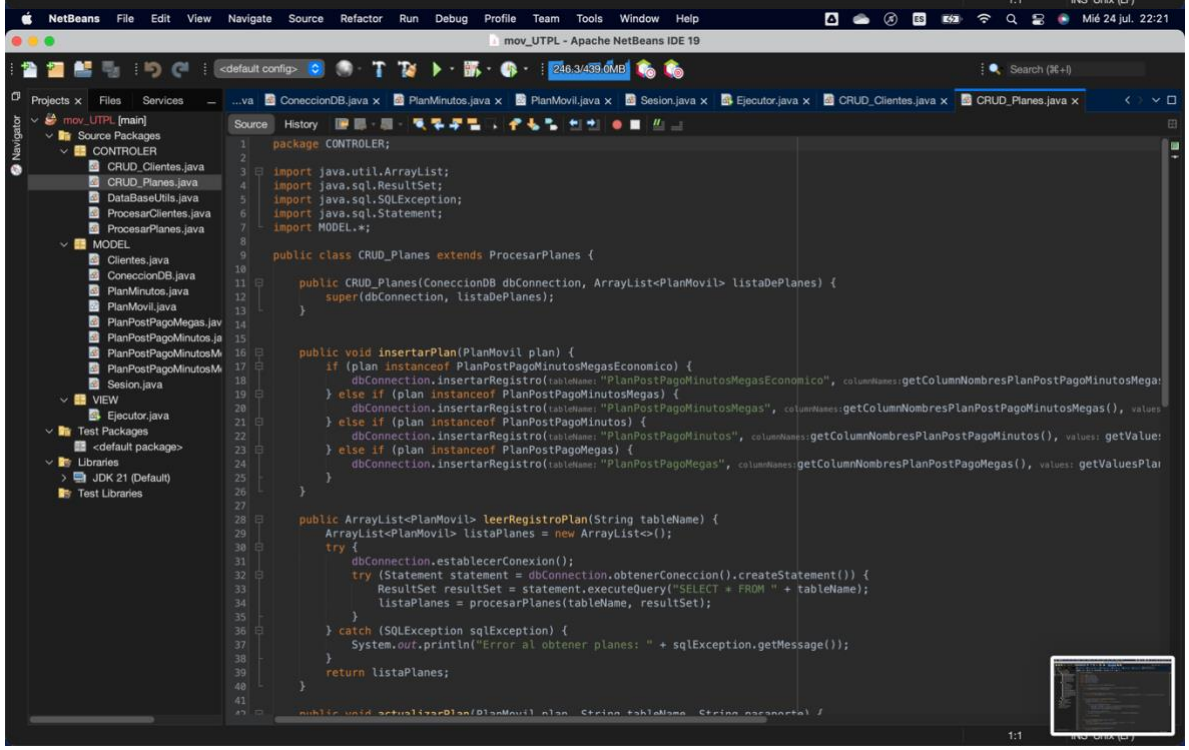

NetBeans File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help
Mié 24 jul. 22:20
mov_UTPL - Apache NetBeans IDE 19
Projects Files Services
Clientes.java ConeccionDB.java PlanMinutos.java
Source History
mov_UTPL [main]
Source Packages
CONTROLER
CRUD_Clientes.java
CRUD_Planes.java
DataBaseUtils.java
ProcesarClientes.java
ProcesarPlanes.java
MODEL
Clientes.java
ConeccionDB.java
PlanMinutos.java
PlanMovil.java
PlanPostPagoMegas.jav
PlanPostPagoMinutos.ja
PlanPostPagoMinutosM
PlanPostPagoMinutosM
Sesion.java
VIEW
Test Packages
<default package>
Libraries
JDK 21 (Default)
Test Libraries
package MODEL;
public class PlanMinutos extends PlanMovil{
public int minutos;
public double costoMinutos;
public PlanMinutos(String pasaporte, int minutos, double costoMinutos, double todos) {
super(pasaporte);
this.minutos = minutos;
this.costoMinutos = costoMinutos;
}
public PlanMinutos(String pasaporte, int minutos) {
super(pasaporte);
this.minutos = minutos;
}
public int getMinutos() {
return minutos;
}
public void setMinutos(int minutos) {
this.minutos = minutos;
}
public double getCostoMinutos() {
return costoMinutos;
}
public void calcularCostoMinutos() {
/*
Precio del minuto de llamada celular en Ecuador
Precio final: $ 0,0448 incluido impuestos
*/
costoMinutos = minutos * 0.0448;
}
@Override
public void calcularPlan() {
throw new UnsupportedOperationException("Not supported yet."); // Generated from nbfs://nbhost/SystemFil
1:1

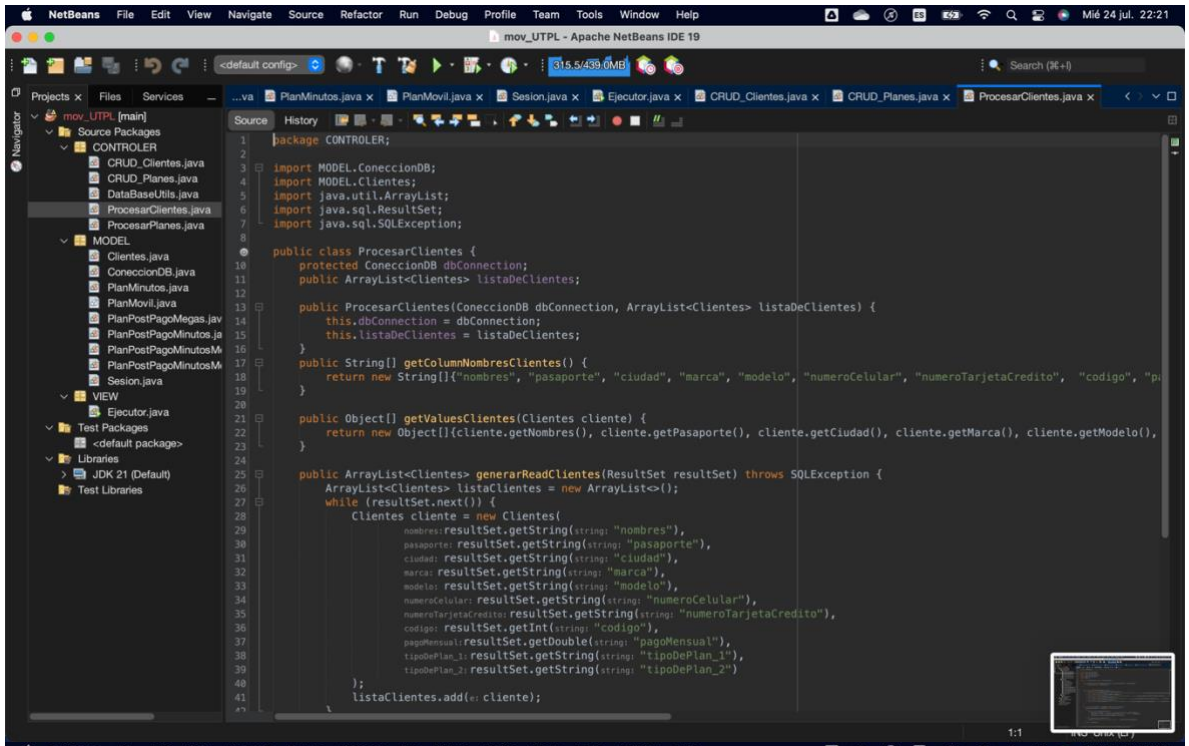

NetBeans File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help
Mié 24 jul. 22:20
mov_UTPL - Apache NetBeans IDE 19
Projects Files Services
Clientes.java ConeccionDB.java PlanMinutos.java PlanMovil.java
Source History
mov_UTPL [main]
Source Packages
CONTROLER
CRUD_Clientes.java
CRUD_Planes.java
DataBaseUtils.java
ProcesarClientes.java
ProcesarPlanes.java
MODEL
Clientes.java
ConeccionDB.java
PlanMinutos.java
PlanMovil.java
PlanPostPagoMegas.jav
PlanPostPagoMinutos.ja
PlanPostPagoMinutosM
PlanPostPagoMinutosM
Sesion.java
VIEW
Test Packages
<default package>
Libraries
JDK 21 (Default)
Test Libraries
package MODEL;
public abstract class PlanMovil {
protected String pasaporte;
protected double total;
public PlanMovil(String pasaporte) {
this.pasaporte = pasaporte;
}
public abstract void calcularPlan();
public abstract String nombrePlan();
public String getPasaporte() {
return pasaporte;
}
public void setPasaporte(String pasaporte) {
this.pasaporte = pasaporte;
}
public double getTotal() {
return total;
}
public void setTotal(double total) {
this.total = total;
}
/*
package MODEL;
public abstract interface PlanMovil {
public abstract void calcularPlan();
public abstract String nombrePlan();
}
*/
}
1:1
INS Unix (LF)

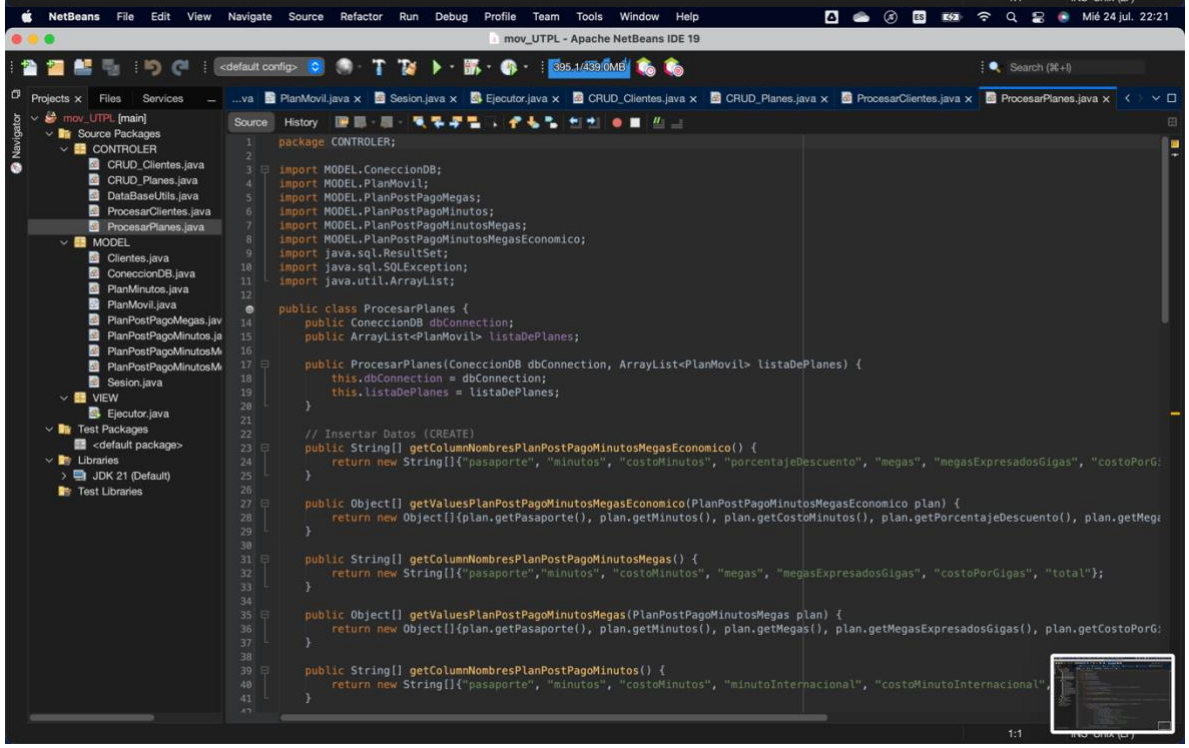

NetBeans File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help
Mié 24 jul. 22:20
mov_UTPL - Apache NetBeans IDE 19
Projects Files Services
Clientes.java ConeccionDB.java PlanMinutos.java PlanMovil.java Sesion.java
Source History
package MODEL;
public class Sesion {
private Clientes cliente;
public void iniciarSesion(Clientes cliente) {
this.cliente = cliente;
}
public void cerrarSesion() {
this.cliente = null;
}
public Clientes getCliente() {
return cliente;
}
public boolean sesionIniciada() {
return cliente != null;
}
}


NetBeans File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help
Mié 24 jul. 22:21
mov_UTPL - Apache NetBeans IDE 19
Projects Files Services
Clientes.java ConeccionDB.java PlanMinutos.java PlanMovil.java Sesion.java Ejecutor.java
Source History
package VIEW;
import MODEL.PlanMovil;
import MODEL.PlanPostPagoMinutosMegasEconomico;
import CONTROLER.*;
import MODEL.*;
import java.util.ArrayList;
import java.util.Scanner;
public class Ejecutor {
public static void main(String[] args) {
Scanner entrada = new Scanner(System.in);
ConeccionDB dbConnection = new ConeccionDB();
ArrayList<Clientes> listaCliente = new ArrayList<>();
ArrayList<PlanMovil> listaPlanes = new ArrayList<>();
CRUD_Planes dataPlanes = new CRUD_Planes(dbConnection, listaPlanes);
CRUD_Clientes dataClientes = new CRUD_Clientes(dbConnection, listaCliente);
Sesion sesion = new Sesion();
String nombre, pasaporte = null, ciudad, marca, modelo, numeroCelular,
numeroTarjetaCredito, TipoDePlan_1, TipoDePlan_2, respuestCompra;
int codigo, minutos, minutoInternacional;
double pagoMensual, megas;
String cadenaOpcion = "Iniciar Sesión";
String cadenaOpcion2 = "Registrarse";
do {
System.out.println("Bienvenido al Sistema de Gestión de clientes y facturación");
System.out.println("1. " + cadenaOpcion);
System.out.println("2. " + cadenaOpcion2);
System.out.println("3. Base de Datos");
System.out.println("4. Generar Factura");
System.out.println("5. Salir");
int opcion = entrada.nextInt();
entrada.nextLine();
switch (opcion) {
case 1:
if (cadenaOpcion.equals("Iniciar Sesión")) {


mov_UTPL - Apache NetBeans IDE 19
Clientes.java
ConeccionDB.java
PlanMinutos.java
PlanMovil.java
Sesion.java
Ejecutor.java
CRUD_Clientes.java
package CONTROLER;
import MODEL.Clientes;
import MODEL.ConeccionDB;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.ArrayList;
public class CRUD_Clientes extends ProcesarClientes {
public CRUD_Clientes(ConeccionDB dbConnection, ArrayList<Clientes> listaDeClientes) {
super(dbConnection, listaDeClientes);
}
public void insertarCliente(Clientes cliente) {
dbConnection.insertarRegistro(tableName: "Clientes", columnNames:getColumnNombresClientes(), values: getValuesClientes(cliente));
}
public ArrayList<Clientes> leerRegistroClientes() {
System.out.println(x: "");
ArrayList<Clientes> listaClientes = new ArrayList<>();
try {
dbConnection.establecerConexion();
try (Statement statement = dbConnection.obtenerConeccion().createStatement()) {
ResultSet resultSet = statement.executeQuery(string: "SELECT * FROM Clientes");
listaClientes = generarReadClientes(resultSet);
}
} catch (SQLException sqlException) {
System.out.println("Error al obtener clientes: " + sqlException.getMessage());
}
return listaClientes;
}
public void actualizarCliente(Clientes cliente) {
System.out.println(x: "");
listaDeClientes.set(index: listaDeClientes.indexOf(o: cliente), element:cliente);
(new ConeccionDB()).actualizarCliente(cliente);
}
public void eliminarCliente(String pasaporte) {
System.out.println(x: "");


mov_UTPL - Apache NetBeans IDE 19
CRUD_Planes.java
package CONTROLER;
import java.util.ArrayList;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import MODEL.*;
public class CRUD_Planes extends ProcesarPlanes {
public CRUD_Planes(ConeccionDB dbConnection, ArrayList<PlanMovil> listaDePlanes) {
super(dbConnection, listaDePlanes);
}
public void insertarPlan(PlanMovil plan) {
if (plan instanceof PlanPostPagoMinutosMegasEconomico) {
dbConnection.insertarRegistro(tableName: "PlanPostPagoMinutosMegasEconomico", columnNames:getColumnNombresPlanPostPagoMinutosMega
} else if (plan instanceof PlanPostPagoMinutosMegas) {
dbConnection.insertarRegistro(tableName: "PlanPostPagoMinutosMegas", columnNames:getColumnNombresPlanPostPagoMinutosMegas(), values
} else if (plan instanceof PlanPostPagoMinutos) {
dbConnection.insertarRegistro(tableName: "PlanPostPagoMinutos", columnNames:getColumnNombresPlanPostPagoMinutos(), values: getValue
} else if (plan instanceof PlanPostPagoMegas) {
dbConnection.insertarRegistro(tableName: "PlanPostPagoMegas", columnNames:getColumnNombresPlanPostPagoMegas(), values: getValuesPla
}
}
public ArrayList<PlanMovil> leerRegistroPlan(String tableName) {
ArrayList<PlanMovil> listaPlanes = new ArrayList<>();
try {
dbConnection.establecerConexion();
try (Statement statement = dbConnection.obtenerConeccion().createStatement()) {
ResultSet resultSet = statement.executeQuery("SELECT * FROM " + tableName);
listaPlanes = procesarPlanes(tableName, resultSet);
}
} catch (SQLException sqlException) {
System.out.println("Error al obtener planes: " + sqlException.getMessage());
}
return listaPlanes;
}


NetBeans File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help
Mié 24 jul. 22:21
mov_UTPL - Apache NetBeans IDE 19
<default config>
315.5/439.0MB
Search (⌘+I)
Projects
Files
Services
mov_UTPL [main]
Source Packages
CONTROLER
CRUD_Clientes.java
CRUD_Planes.java
DataBaseUtils.java
ProcesarClientes.java
ProcesarPlanes.java
MODEL
Clientes.java
ConeccionDB.java
PlanMinutos.java
PlanMovil.java
PlanPostPagoMegas.jav
PlanPostPagoMinutos.ja
PlanPostPagoMinutosM
PlanPostPagoMinutosM
Sesion.java
VIEW
Ejecutor.java
Test Packages
<default package>
Libraries
JDK 21 (Default)
Test Libraries
PlanMinutos.java
PlanMovil.java
Sesion.java
Ejecutor.java
CRUD_Clientes.java
CRUD_Planes.java
ProcesarClientes.java
Source
History
1:1


NetBeans File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help
Mié 24 jul. 22:21
mov_UTPL - Apache NetBeans IDE 19
<default config>
395.1/439.0MB
Search (⌘+I)
Projects
Files
Services
mov_UTPL [main]
Source Packages
CONTROLER
CRUD_Clientes.java
CRUD_Planes.java
DataBaseUtils.java
ProcesarClientes.java
ProcesarPlanes.java
MODEL
Clientes.java
ConeccionDB.java
PlanMinutos.java
PlanMovil.java
PlanPostPagoMegas.jav
PlanPostPagoMinutos.ja
PlanPostPagoMinutosM
PlanPostPagoMinutosM
Sesion.java
VIEW
Ejecutor.java
Test Packages
<default package>
Libraries
JDK 21 (Default)
Test Libraries
PlanMovil.java
Sesion.java
Ejecutor.java
CRUD_Clientes.java
CRUD_Planes.java
ProcesarClientes.java
ProcesarPlanes.java
Source
History
1:1

4. Resultados