Universidad Técnica Particular de Loja

PROGRAMACION ORIENTADA A OBJETOS

Ing. Wayner Bustamante

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

Jean Álvarez / 1105977225

Ruben Condoy / 1106044553

Cristian Jiménez / 1150748182

27/07/2023

Loja - Ecuador

Contenido:

GESTION DE TELEFONIA MOVIL ESTUDIANTIL: Mov-UTPL

1. **OBJETIVOS:**

- Gestión de telefonía móvil estudiantil: Mov-UTPL.
- Aplicación los pilares de la POO: herencia y polimorfismo, generando resultados desde/hacia DB.

2. **MATERIALES Y PROGRAMAS:**

- APACHE NETBEANS
- IntelliJ Idea
- MySqlLite
- Browser my sqlLite
- Dia
- UML
- JAVA

3. ANÁLISIS.

Análisis de la problemática:

El problema planteado se refiere a la creación de un "Sistema de Gestión de clientes y facturación Mov-UTPL" para una carrera de Telecomunicación en la UTPL (Universidad Técnica Particular de Loja) que busca impulsar un proyecto de comunicación móvil para su comunidad universitaria. La infraestructura tecnológica ya está disponible, pero el sistema requiere un software que gestione la información de los clientes y la facturación de sus planes móviles.

Requisitos y características del sistema:

Datos básicos del cliente: Se debe permitir el registro y mantenimiento de información del cliente, incluyendo nombres, pasaporte/cédula, ciudad, marca, modelo y número de celular, además de al menos 2 atributos adicionales, para los cuales seleccionamos la edad y el correo electrónico del cliente.

Tipos de planes móviles: Se deben implementar varios tipos de planes con características específicas:

- PlanPostPagoMinutosMegasEconomico: Almacena información sobre minutos, costo por minuto, megas en gigas, costo por giga y un porcentaje de descuento.
- PlanPostPagoMinutos: Contiene datos sobre minutos nacionales, costo por minuto nacional, minutos internacionales y costo por minuto internacional.

- PlanPostPagoMegas: Tiene información sobre megas en gigas, costo por giga y tarifa base.
- PlanPostPagoMinutosMegas: Incluye datos de minutos, costo por minuto, megas en gigas y costo por giga.

Operaciones CRUD: Se debe facilitar la gestión de clientes y planes mediante operaciones CRUD (Create, Read, Update, Delete), permitiendo agregar, consultar, actualizar y eliminar registros.

Generación de facturas: El sistema debe generar facturas para cada cliente y su respectivo plan, aplicando las tarifas y descuentos correspondientes según el tipo de plan.

Restricción de tipos de planes: Cada cliente/estudiante solo puede disponer de un máximo de 2 tipos de planes.

Motor de Base de Datos (DB): Se debe utilizar SQLite como motor de base de datos para almacenar la información de clientes, planes y facturas.

Documentación y diseño en UML: Es necesario documentar de manera oportuna el análisis y diseño del sistema utilizando UML (Unified Modeling Language) para representar de manera clara y eficiente la estructura del sistema, aplicando herencia y polimorfismo donde corresponda.

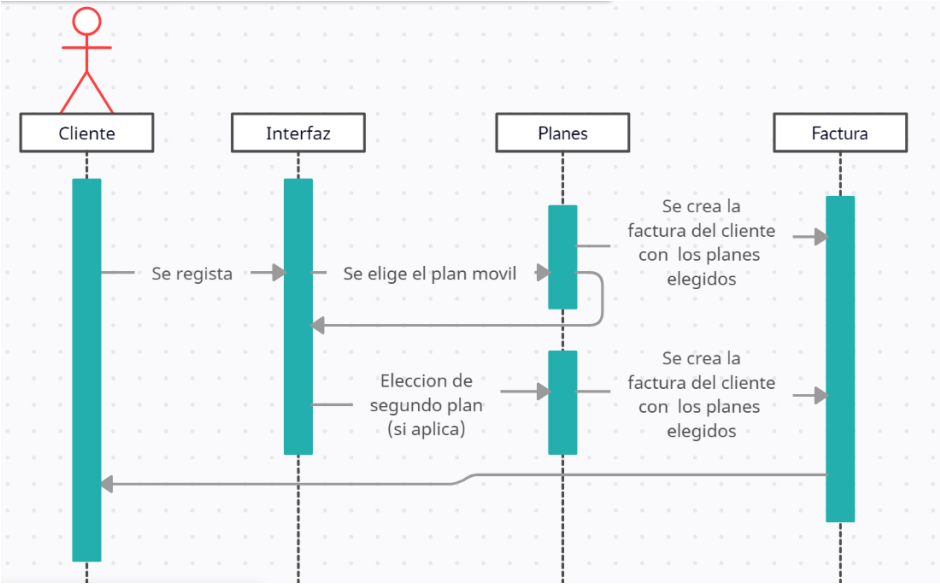
Arquitectura MVC: Se debe aplicar la arquitectura Modelo-Vista-Controlador (MVC), asegurando que la lógica de negocio, la presentación y la interacción con la base de datos estén separadas adecuadamente.

Solución propuesta

Para abordar la problemática y dar solución al problema planteado, se propone el desarrollo de una aplicación de gestión móvil estudiantil basada en los principios de Programación

Orientada a Objetos (POO) utilizando el lenguaje de programación adecuado en nuestro caso será en Java.

- Análisis y Diseño UML: Se llevará a cabo un análisis detallado de los requisitos y una representación visual del diseño utilizando diagramas UML, incluyendo diagramas de

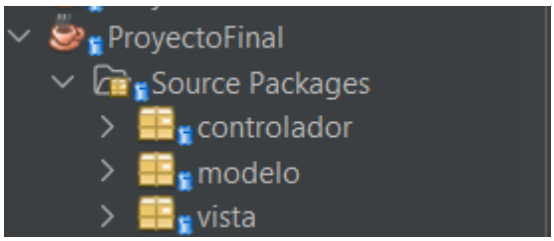


clases para los clientes y los diferentes tipos de planes móviles, así como diagramas de secuencia para las operaciones CRUD y la generación de facturas.

- Implementación de Clases con Herencia y Polimorfismo: Se crearán clases para representar a los clientes y a los diferentes tipos de planes móviles, aplicando el concepto de herencia para aprovechar la similitud entre los diferentes planes. Se utilizará el polimorfismo para permitir que diferentes tipos de planes sean tratados de manera uniforme en ciertas operaciones, lo que simplificará el código y lo hará más flexible.

Atributos	minutos	costoMinutos	megasEnGigas	costoPorGiga	porcentajeDescuento	Tarifa base	minutosInternacionales	costoMinutosInternacio
PlanPostPagoMinutosMegasEconomico	x	x	x	x	x	x		
PlanPostPagoMinutosMegas	x	x	x	x				
PlanPostPagoMinutos	x	x					x	x
PlanPostPagoMegas			x	x		x		
Atributos comunes	~	~	~	~				

- Aplicación de Arquitectura MVC: Se organizará la aplicación siguiendo la arquitectura Modelo-Vista-Controlador (MVC), separando la lógica de negocio, la presentación y la interacción con la base de datos.



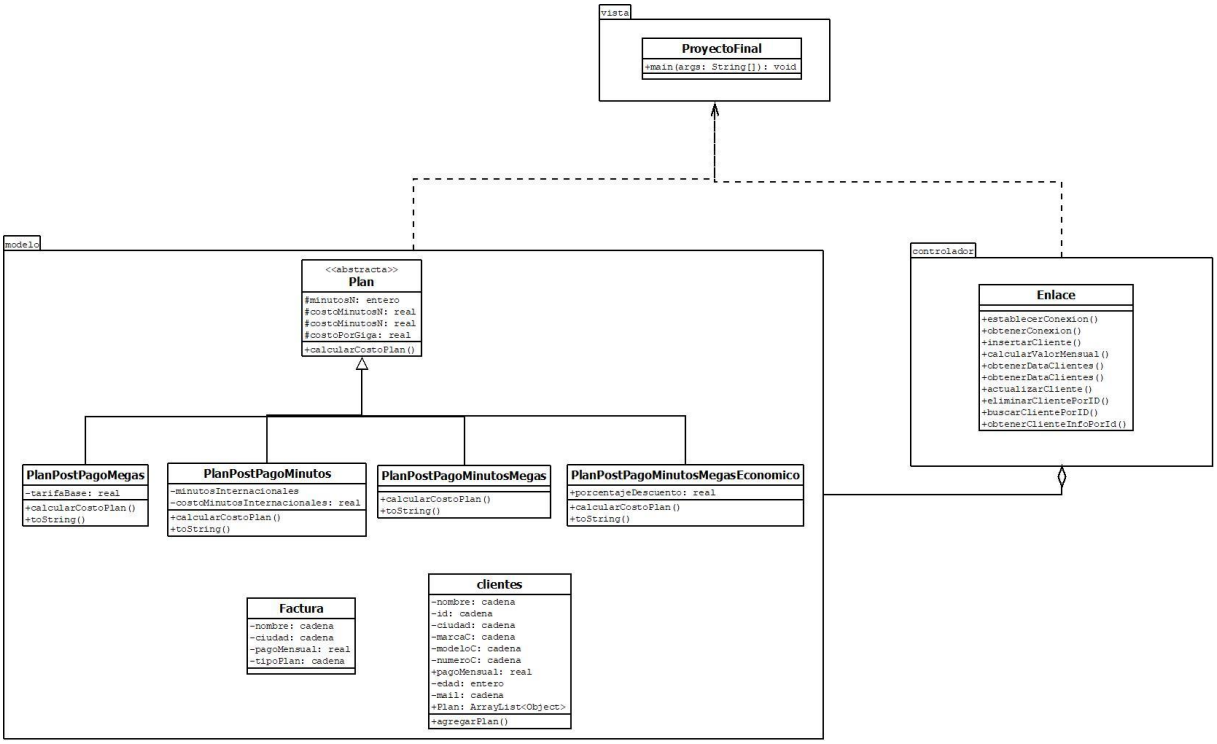
- Gestión de Base de Datos: Se implementará la lógica necesaria para gestionar la base de datos SQLite, asegurando que se puedan realizar las operaciones CRUD y almacenar la información de clientes, planes y facturas.

- Generación de Facturas: Se desarrollará un mecanismo para generar las facturas de cada cliente en función de su plan y los cargos correspondientes, teniendo en cuenta los descuentos aplicables.
- Pruebas y Depuración: Se realizarán pruebas exhaustivas del sistema para asegurar su correcto funcionamiento y se corregirán posibles errores o fallos identificados durante esta fase.
- Documentación del Proceso: Se documentarán todos los pasos del proceso de desarrollo, desde el análisis hasta la implementación, incluyendo las decisiones de diseño tomadas y los resultados de las pruebas realizadas.

La solución propuesta busca crear un sistema eficiente, flexible y fácil de mantener que permita a la comunidad universitaria de la UTPL gestionar sus planes móviles de manera efectiva y brinde una experiencia óptima a los usuarios.

Finalmente, luego de analizar algunas maneras de darle solución al problema, decidimos empezar a codificar clase por clase, empezando con el enlace de la base de datos y las clases de los planes que heredarían de una clase padre los atributos que consideramos pertinentes y el método abstracto que permitía calcular el costo de cada plan, luego poco a poco empezaríamos a brindar las demás características CRUD que nos faltaban, además de agregar la interfaz que se imprimiría en pantalla y nos permitiría interactuar con el programa

4. DISEÑO.



5. CODIFICACIÓN.

La parte más importante del código es la implementación del menú principal en el método main(). En este método, se muestra un menú interactivo que permite a los usuarios realizar diversas acciones relacionadas con la gestión de clientes y sus planes. Esto es crucial porque brinda una interfaz fácil de usar para que los usuarios interactúen con el sistema y realicen las operaciones que necesiten.

En resumen, la implementación del menú principal permite a los usuarios:

Crear nuevos clientes: El usuario ingresa los datos de un nuevo cliente, como nombre, ID, ciudad, marca y modelo del celular, número de teléfono, edad y correo electrónico.

Luego, se le solicita que seleccione el tipo de plan que desea para el cliente, ya sea un plan de minutos, un plan de megas, un plan combinado de minutos y megas, o un plan combinado de minutos, megas y descuento.

Puede agregar uno o dos planes al cliente, según su elección.

Mostrar todos los clientes: Muestra una lista de todos los clientes registrados en el sistema, mostrando información como nombre, ID y ciudad.

Actualizar datos de un cliente: El usuario ingresa el ID del cliente que desea actualizar. Luego, puede modificar los datos del cliente, como nombre, ciudad, marca y modelo del celular, número de teléfono, edad y correo electrónico.

Eliminar un cliente: El usuario ingresa el ID del cliente que desea eliminar.

El cliente correspondiente se elimina de la base de datos.

Mostrar factura de un cliente: El usuario ingresa el ID del cliente del cual desea obtener la factura. La factura muestra información sobre el cliente, como nombre, ciudad, pago mensual y tipo de plan(es) asociado(s).

La implementación del menú principal es fundamental para que el programa sea interactivo y permita a los usuarios realizar todas estas acciones sin tener que trabajar directamente con la base de datos o la lógica interna del sistema. En lugar de eso, pueden interactuar con el sistema a través de una interfaz intuitiva y amigable. Esto mejora significativamente la usabilidad y la experiencia del usuario al utilizar la aplicación.

La clase "Enlace" también es importante en el proyecto, ya que actúa como una capa de conexión entre la lógica de negocio (las clases "clientes" y "Factura") y la capa de presentación (la clase "ProyectoFinal" que contiene el menú y la interacción con el usuario). La clase "Enlace" proporciona métodos para realizar operaciones de inserción, búsqueda, actualización y eliminación de datos de clientes, además de obtener información relevante para generar la factura.

```java
package vista;

    import controlador.Enlace;
    import modelo.*;

    import java.util.ArrayList;
    import java.util.List;
    import java.util.Scanner;

    /**
     *
     * @author Jeanca and Ruben
     */
    public class ProyectoFinal {

        /**
         * @param args the command line arguments
         */
        public static void main(String[] args) {
            Enlace enlace = new Enlace();
            Scanner scanner = new Scanner(System.in);
            int opcion;

            do {
                System.out.println("\n—— Menú ——");
                System.out.println("1. Crear nuevo cliente");
                System.out.println("2. Mostrar todos los clientes");
                System.out.println("3. Actualizar datos de un cliente");
                System.out.println("4. Eliminar un cliente");
                System.out.println("5. Mostrar factura de un cliente");
                System.out.println("0. Salir");
                System.out.print("Seleccione una opción: ");
                opcion = scanner.nextInt();
                scanner.nextLine();

                switch (opcion) {
                    case 1:
                        // Crear un nuevo cliente
                        clientes nuevoCliente = obtenerDatosPorTeclado(scanner);

                        // Preguntar al cliente qué tipo de plan desea
                        System.out.println("\nSeleccione el tipo de plan para el primer plan:");
                        System.out.println("1. Plan PostPago Minutos");
                        System.out.println("2. Plan PostPago Megas");
                        System.out.println("3. Plan PostPago Minutos y Megas");
                        System.out.println("4. Plan PostPago Minutos, Megas y Descuento");
                        int opcionPlan1 = scanner.nextInt();
                        scanner.nextLine();
```

```java
// Crear el primer plan
                        Plan plan1 = crearPlan(opcionPlan1, scanner);
                        if (plan1 != null) {
                            nuevoCliente.agregarPlan(plan1);
                        }

// Preguntar si desea agregar un segundo plan
                        scanner.nextLine();
                        System.out.println("¿Desea agregar otro plan? (S/N)");
                        String respuesta = scanner.nextLine();
                        if (respuesta.equalsIgnoreCase("S")) {
                            System.out.println("\nSeleccione el tipo de plan para el segundo plan:");
                            System.out.println("1. Plan PostPago Minutos");
                            System.out.println("2. Plan PostPago Megas");
                            System.out.println("3. Plan PostPago Minutos y Megas");
                            System.out.println("4. Plan PostPago Minutos, Megas y Descuento");
                            int opcionPlan2 = scanner.nextInt();
                            scanner.nextLine();

                            Plan plan2 = crearPlan(opcionPlan2, scanner);
                            if (plan2 != null) {
                                nuevoCliente.agregarPlan(plan2);
                            }
                        }

                        enlace.insertarCliente(nuevoCliente);
                        System.out.println("¡Nuevo cliente creado exitosamente!");
                        break;
                    case 2:
                        // Mostrar todos los clientes
                        ArrayList<clientes> listaClientes = enlace.obtenerDataClientes();
                        mostrarClientes(listaClientes);
                        break;
                    case 3:
                        // Actualizar datos de un cliente
                        System.out.print("Ingrese el ID del cliente que desea actualizar: ");
                        String idCliente = scanner.nextLine();
                        clientes clienteExistente = enlace.buscarClientePorID(idCliente);
                        if (clienteExistente != null) {
                            actualizarClientePorTeclado(clienteExistente, scanner);
                            enlace.actualizarCliente(clienteExistente);
                            System.out.println("Datos del cliente actualizados exitosamente.");
                        } else {
                            System.out.println("Cliente no encontrado.");
                        }
                        break;
```

```java
                    case 4:
                        // Eliminar un cliente
                        System.out.print("Ingrese el ID del cliente que desea eliminar: ");
                        String idClienteAEliminar = scanner.nextLine();
                        enlace.eliminarClientePorID(idClienteAEliminar);
                        System.out.println("Cliente eliminado exitosamente.");
                        break;
                    case 5:
                        // Mostrar factura de un cliente
                        System.out.print("Ingrese el ID del cliente para ver la factura: ");
                        String idClienteFactura = scanner.nextLine();
                        Factura clienteFactura = enlace.obtenerClienteInfoPorId(idClienteFactura);
                        if (clienteFactura != null) {
                            System.out.println("Factura del cliente con ID " + idClienteFactura + ":");
                            System.out.println("Nombre: " + clienteFactura.getNombre());
                            System.out.println("Ciudad: " + clienteFactura.getCiudad());
                            System.out.println("Pago Mensual: " + clienteFactura.getPagoMensual());
                            System.out.println("Tipo de Plan(es): " + clienteFactura.getTipoPlan());
                        } else {
                            System.out.println("No se encontró ningún cliente con el ID " +
idClienteFactura);
                        }
                            break;
                    case 0:
                        System.out.println("Saliendo del programa...");
                        break;

                    default:
                        System.out.println("Opción inválida. Intente nuevamente.");
                }

            } while (opcion != 0);

            scanner.close();
        }

        // Método para obtener los datos de un nuevo cliente por teclado
        public static clientes obtenerDatosPorTeclado(Scanner scanner) {
            clientes nuevoCliente = new clientes();

            System.out.println("Ingrese los datos del nuevo cliente:");
            System.out.print("Nombre: ");
            nuevoCliente.setNombre(scanner.nextLine());

            System.out.print("ID: ");
            nuevoCliente.setId(scanner.nextLine());

            System.out.print("Ciudad: ");
            nuevoCliente.setCiudad(scanner.nextLine());

            System.out.print("Marca del celular: ");
            nuevoCliente.setMarcaC(scanner.nextLine());

            System.out.print("Modelo del celular: ");
            nuevoCliente.setModeloC(scanner.nextLine());

            System.out.print("Número de teléfono: ");
            nuevoCliente.setNumeroC(scanner.nextLine());

            System.out.print("Edad: ");
            nuevoCliente.setEdad(scanner.nextInt());
            scanner.nextLine(); // Consume la nueva línea pendiente después del nextInt()

            System.out.print("Correo electrónico: ");
            nuevoCliente.setMail(scanner.nextLine());

            return nuevoCliente;
        }

        // Método para mostrar los datos de todos los clientes
        public static void mostrarClientes(ArrayList<clientes> listaClientes) {
            System.out.println("\nClientes registrados:");
            for (clientes cliente : listaClientes) {
                System.out.println("Nombre: " + cliente.getNombre());
                System.out.println("ID: " + cliente.getId());
                System.out.println("Ciudad: " + cliente.getCiudad());
                System.out.println("-----------------------");
            }
        }

        // Método para actualizar los datos de un cliente por teclado
        public static void actualizarClientePorTeclado(clientes cliente, Scanner scanner) {

            System.out.println("Ingrese los nuevos datos del cliente:");
            System.out.print("Nombre: ");
            cliente.setNombre(scanner.nextLine());

            System.out.print("Ciudad: ");
            cliente.setCiudad(scanner.nextLine());

            System.out.print("Marca del celular: ");
            cliente.setMarcaC(scanner.nextLine());

            System.out.print("Modelo del celular: ");
            cliente.setModeloC(scanner.nextLine());

            System.out.print("Número de teléfono: ");
            cliente.setNumeroC(scanner.nextLine());

            System.out.print("Edad: ");
            cliente.setEdad(scanner.nextInt());
            scanner.nextLine();

            System.out.print("Correo electrónico: ");
            cliente.setMail(scanner.nextLine());
        }
```

```java
// Método para crear un plan según la opción seleccionada por el usuario
        public static Plan crearPlan(int opcionPlan, Scanner scanner) {
            Plan plan = null;
            switch (opcionPlan) {
                case 1:
                    System.out.println("Ingrese la cantidad de minutos internacionales:");
                    int minutosInternacionales = scanner.nextInt();
                    System.out.println("Ingrese el costo de minutos internacionales:");
                    double costoMinutosInternacionales = scanner.nextDouble();
                    int megasGigas = 0;
                    double costoMegasGigas = 0;
                    System.out.print("Ingrese la cantidad de minutos: ");
                    int minutosN = scanner.nextInt();
                    System.out.print("Ingrese el costo por minuto: ");
                    double costoMinutosN = scanner.nextDouble();
                    scanner.nextLine();
                    plan = new PlanPostPagoMinutos( minutosInternacionales,
costoMinutosInternacionales, minutosN, costoMinutosN, megasGigas, costoMegasGigas);
                    break;
                case 2:
                    System.out.print("Ingrese la cantidad de megas: ");
                    double megasAGigas = scanner.nextDouble();
                    System.out.print("Ingrese el costo por giga: ");
                    double costoPorGiga = scanner.nextDouble();
                    System.out.print("Ingrese la tarifa base: ");
                    double tarifaBase = scanner.nextDouble();
                    int minutosN2 = 0;
                    double costoMinutosN2 = 0;
                    plan = new PlanPostPagoMegas(tarifaBase, minutosN2, costoMinutosN2, megasAGigas,
costoPorGiga);
                    break;
                case 3:
                    System.out.print("Ingrese la cantidad de minutos: ");
                    int minutosNM = scanner.nextInt();
                    System.out.print("Ingrese el costo por minuto: ");
                    double costoMinutosNM = scanner.nextDouble();
                    System.out.print("Ingrese la cantidad de megas: ");
                    double megasAGigasNM = scanner.nextDouble();
                    System.out.print("Ingrese el costo por giga: ");
                    double costoPorGigaNM = scanner.nextDouble();
                    scanner.nextLine();
                    plan = new PlanPostPagoMinutosMegas(minutosNM, costoMinutosNM, megasAGigasNM,
costoPorGigaNM);
                    break;
                case 4:
                    System.out.print("Ingrese la cantidad de minutos: ");
                    int minutosNMD = scanner.nextInt();
                    System.out.print("Ingrese el costo por minuto: ");
                    double costoMinutosNMD = scanner.nextDouble();
                    System.out.print("Ingrese la cantidad de megas: ");
                    double megasAGigasNMD = scanner.nextDouble();
                    System.out.print("Ingrese el costo por giga: ");
                    double costoPorGigaNMD = scanner.nextDouble();
                    System.out.print("Ingrese el porcentaje de descuento: ");
                    double porcentajeDescuento = scanner.nextDouble();
                    scanner.nextLine();
                    plan = new PlanPostPagoMinutosMegasEconomico(porcentajeDescuento, minutosNMD,
costoMinutosNMD, megasAGigasNMD, costoPorGigaNMD);
                    break;
                default:
                    System.out.println("Opción inválida. No se agregará ningún plan al cliente.");
            }
            return plan;
        }
    }
```

URL Git del proyecto:

https://github.com/ProOrientadaObjetos-P-D-AA2023/proyecto-final-JeanAlvarez21/blob/main/2_SOLUCION/ProyectoFinal/src/vista/ProyectoFinal.java

- RESULTADOS.

  Resultados de la codificación del programa en NetBeans.

Al ejecutar el programa, ejecuta lo siguiente:

```
--- Menu ---
1. Crear nuevo cliente
2. Mostrar todos los clientes
3. Actualizar datos de un cliente
4. Eliminar un cliente
5. Mostrar factura de un cliente
0. Salir
Seleccione una opcion: 1
Ingrese los datos del nuevo cliente:
Nombre: Jean
ID: 12335
Ciudad: Loja
Marca del celular: Iphone 15
Modelo del celular: IPhone
N◆mero de tel◆fono: 0586468484
Edad: 45
Correo electr◆nico: jeanalvarez@gmail.com

Seleccione el tipo de plan para el primer plan:
1. Plan PostPago Minutos
2. Plan PostPago Megas
3. Plan PostPago Minutos y Megas
4. Plan PostPago Minutos, Megas y Descuento
1
Ingrese la cantidad de minutos internacionales:
12


Ingrese el costo de minutos internacionales:
0.4
Ingrese la cantidad de minutos: 12
Ingrese el costo por minuto: 1
2
◆Desea agregar otro plan? (S/N)
N
◆Nuevo cliente creado exitosamente!

--- Menu ---
1. Crear nuevo cliente
2. Mostrar todos los clientes
3. Actualizar datos de un cliente
4. Eliminar un cliente
5. Mostrar factura de un cliente
0. Salir
Seleccione una opcion: 2
Exception: no such column: 'minutosN'

Clientes registrados:
Nombre: Jean
ID: 1105977225
Ciudad: Loja
--------------------
Nombre: Jean
ID: 12335
Ciudad: Loja
```

```
--- Menu ---
1. Crear nuevo cliente
2. Mostrar todos los clientes
3. Actualizar datos de un cliente
4. Eliminar un cliente
5. Mostrar factura de un cliente
0. Salir
Seleccione una opcion: 5
Ingrese el ID del cliente para ver la factura: 12335
Factura del cliente con ID 12335:
Nombre: Jean
Ciudad: Loja
Pago Mensual: 16.8
Tipo de Plan(es): PlanPostPagoMinutos

--- Menu ---
1. Crear nuevo cliente
2. Mostrar todos los clientes
3. Actualizar datos de un cliente
4. Eliminar un cliente
5. Mostrar factura de un cliente
0. Salir
Seleccione una opcion: 0
Saliendo del programa...
BUILD SUCCESSFUL (total time: 2 minutes 23 seconds)
```