

**UNIVERSIDAD TÉCNICA PARTICULAR DE LOJA**

**Informe Proyecto Final POO**

Paul Espinosa

Pablo Mendoza

II Ciclo de Ing. En Ciencias de la Computacion

Programación Orientada a Objetos

Ing. Pedro Daniel Irene Robalino

03 de aug. de 2023

# PROYECTO DE GRADO

Contenido

[PROYECTO DE GRADO 2](#_Toc141988080)

[Análisis de la Solución 3](#_Toc141988081)

[Diseño 7](#_Toc141988082)

[Codificación 12](#_Toc141988083)

[Resultados 18](#_Toc141988084)

## Análisis de la Solución

Para abordar este problema, se deben considerar los siguientes elementos:

Datos básicos del cliente:

Se requiere almacenar información personal de los clientes, como nombres, pasaporte/cédula, ciudad de residencia, marca y modelo del celular, número de celular y el pago mensual. Además, se solicita agregar al menos 2 atributos adicionales. Estos atributos adicionales no se detallan en la problemática, por lo que sería necesario definirlos con mayor precisión en la implementación del sistema.

Tipos de planes móviles:

Existen diferentes tipos de planes que se ofrecen a los clientes. Estos planes se clasifican en cuatro categorías:

a) PlanPostPagoMinutosMegasEconomico: Este plan debe almacenar información sobre la cantidad de minutos, el costo por minuto, la cantidad de megas expresados en gigas, el costo por cada giga y el porcentaje de descuento aplicable.

b) PlanPostPagoMinutos: En este tipo de plan se deben registrar los minutos nacionales e internacionales, junto con sus respectivos costos por minuto. Es importante definir cómo se calculará el costo total del plan, ya que pueden existir diferentes combinaciones de minutos nacionales e internacionales.

c) PlanPostPagoMegas: Para este plan, se necesita almacenar la cantidad de megas expresados en gigas, el costo por cada giga y una tarifa base. Al igual que en los otros planes, es fundamental tener un criterio claro para calcular el costo total del plan.

d) PlanPostPagoMinutosMegas: Este plan combina características de minutos y megas. Se debe registrar la cantidad de minutos, el costo por minuto, la cantidad de megas expresados en gigas y el costo por cada giga. Es importante definir cómo se determinará el costo total del plan considerando ambas variables.

Dado que la información de los planes varía según su tipo, es posible crear tablas para cada tipo de plan y definir los campos necesarios para almacenar los datos correspondientes:

1. PlanPostPagoMinutosMegasEconomico

|  |  |
| --- | --- |
| Atributo | Descripción |
| Minutos | Cantidad de minutos incluidos |
| CostoMinutos | Precio por minuto adicional |
| Megas (en gigas) | Cantidad de megas incluidos |
| CostoPorGiga | Precio por cada giga adicional |
| PorcentajeDescuento | Porcentaje de descuento applicable |

1. PlanPostPagoMinutos

|  |  |
| --- | --- |
| Atributo | Descripción |
| MinutosNacionales | Cantidad de minutos nacionales |
| CostoMinutoNacional | Precio por minuto nacional |
| MinutosInternacionales | Cantidad de minutos internacionales |
| CostoMinutoInternacional | Precio por minuto internacional |

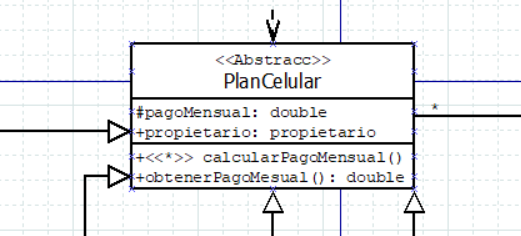
1. PlanPostPagoMegas

|  |  |
| --- | --- |
| Atributo | Descripción |
| Megas (en gigas) | Cantidad de megas incluidos |
| CostoPorGiga | Precio por cada giga adicional |
| TarifaBase | Tarifa base del plan |

1. PlanPostPagoMinutosMegas

|  |  |
| --- | --- |
| Atributo | Descripción |
| Minutos | Cantidad de minutos incluidos |
| CostoMinutos | Precio por minuto adicional |
| Megas (en gigas) | Cantidad de megas incluidos |
| CostoPorGiga | Precio por cada giga adicional |

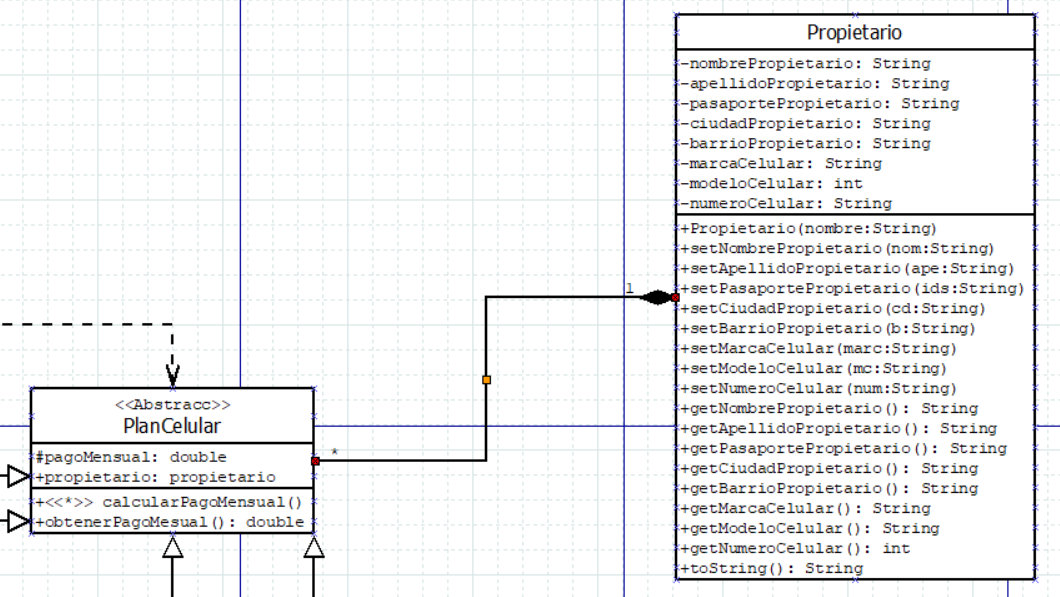
## Diseño



Aquí se crea la clase abstracta PlanCelular que será la clase padre de las diferentes clases de planes, al ser abstracta poseerá métodos abstractos, los cuales son:

CalcularPagoMensual y ObtenerPagoMensual

Luego se crea un propietario de tipo propietario, con un enlace de relación tipo composición, el cual unirá a las clases PlanCelular y Propietario:

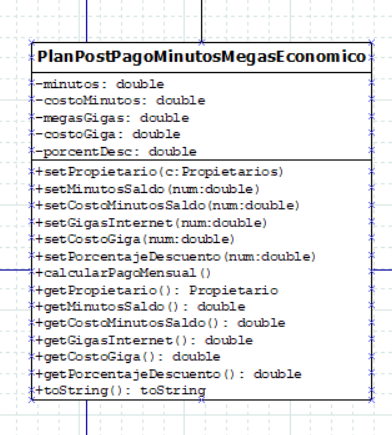


En la clase propietario se definirán los atributos(nombre, apellido, pasaporte, ciudad, barrio, marcaCelular, modeloCelular y numeroCelular) que representaran los datos básicos del propietario.

Luego de esto viene la creación de las clases hijas o herederas de la clase PlanCelular:

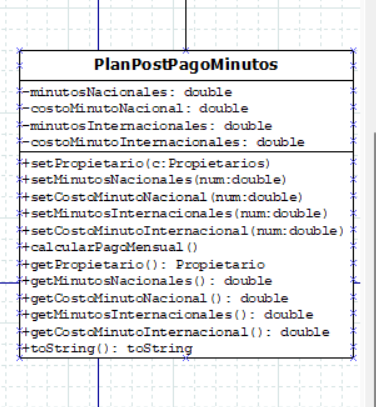
PlanPostPagoMinutosMegasEconomico

Quienes deben almacenar la siguiente información: minutos, costo minutos, megas expresados en gigas, costo por cada giga, porcentaje de descuento.



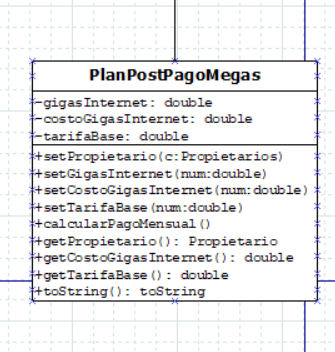
PlanPostPagoMinutos

Con los siguientes datos: minutos nacionales, costo minuto nacional, minutos internacionales, costo minuto internacional.



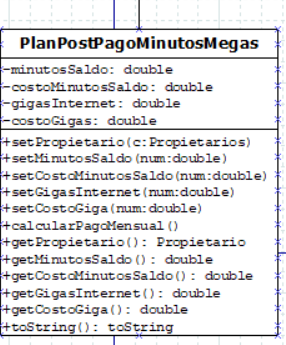
PlanPostPagoMegas

La información para este tipo de plan es: megas expresados en gigas, costo por cada giga, tarifa base.

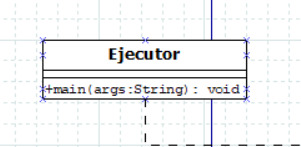


PlanPostPagoMinutosMegas

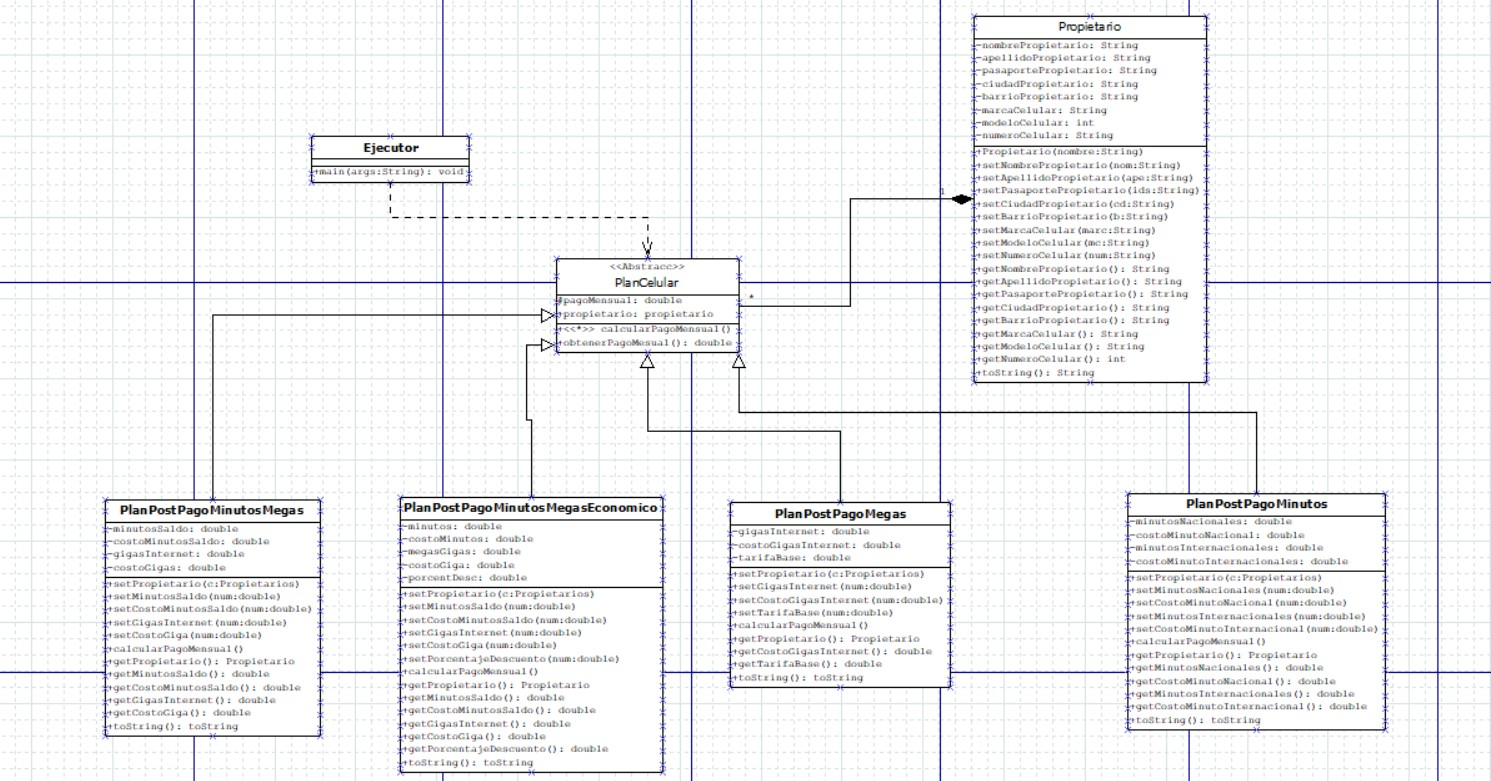
Con datos como: minutos, costo minutos, megas expresados en gigas, costo por cada giga.



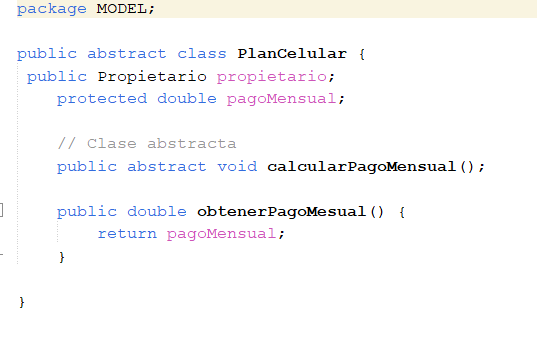
Al final se agregaría el ejecutor, el cual se define de la siguiente manera:



Siendo el UML final, tomando en cuenta enlaces necesarios y clases con sus respectivos atributos y métodos definidos:



## Codificación

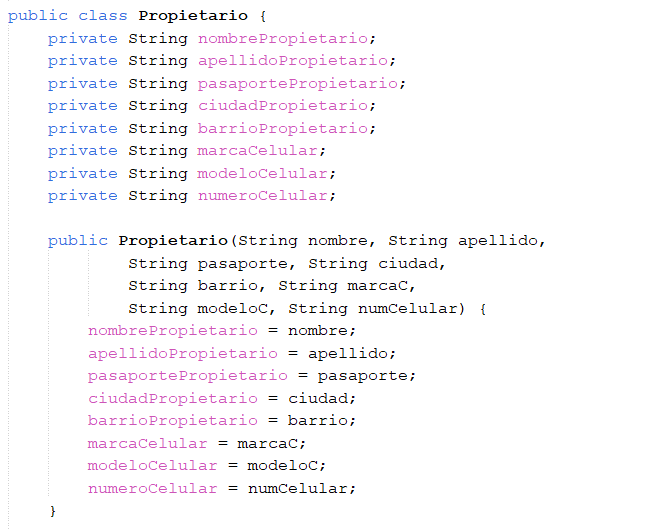


Aquí se crea la clase abstracta PlanCelular que será la clase padre de las diferentes clases de planes, esta posee un método abstracto estos son:

* public abstract void calcularPagoMensual(): este método será implementado de manera diferente en todas las clase, dependiendo de su método especifico.
* public double obtenerPagoMesual(): esto nos ayudara a obtener el pago mensual en cada clase.

Aclaramos que este sería tanto el uso de la herencia y del polimorfismo, ya que el polimorfismo no puede existir sin la herencia.

Por lo mencionado anteriormente esta clase tendrá un atributo tipo propietario, lo cual nos ayuda a que todos los datos del propietario pasen tanto a la clase PlanCelular como a sus clases hijas.



Esta es la clase Propietario mencionada anteriormente y como se muestra en las imágenes esta posee todos los datos que se necesitan del propietario.



Esta es la clase que enlaza la base de datos al programa, y lo cual permitirá el uso de los datos dentro de esta para realizar los diferentes procesos.



Este es el código mediante el cual se insertan los planes en la base de datos, debidamente en la tabla con el mismo nombre, este proceso se repite para todos los planes.

Debido a que es un código que se usa en varias ocaciones lo explicaremos en mas detalle usando uno de los planes.

1. public void insertarPlanPostPagoMinutosMegasEconomico(PlanPostPagoMinutosMegasEconomico p): Este es el encabezado del método. Está definido como público (puede ser accedido desde fuera de la clase), no devuelve ningún valor (void), y toma un parámetro de tipo `PlanPostPagoMinutosMegasEconomico` llamado "p". Presumiblemente, esta clase `PlanPostPagoMinutosMegasEconomico` contiene información sobre el plan de postpago a insertar.
2. try { ... } catch (SQLException e) { ... }: Aquí se inicia un bloque de código "try-catch", que maneja excepciones en caso de que ocurra algún error durante la ejecución del código dentro del bloque "try". En este caso, se espera que las excepciones de tipo `SQLException` sean capturadas y manejadas.
3. establecerConexion(): Este es un método (que aparentemente no se muestra en el código) que se supone establece la conexión con la base de datos.
4. Statement statement = obtenerConexion().createStatement(): Se crea un objeto `Statement` que se utilizará para ejecutar consultas SQL en la base de datos. Este objeto se crea a partir de la conexión obtenida mediante `obtenerConexion()`.
5. String data = String.format("INSERT INTO ...): Aquí se construye una cadena de consulta SQL utilizando los valores del objeto `p` (instancia de `PlanPostPagoMinutosMegasEconomico`). Esta consulta SQL es un comando de inserción que agrega nuevos datos a la tabla "planPostPagoMinutosMegasEconomico". Los valores a insertar son tomados de diferentes métodos del objeto `p`, como el nombre del propietario, el número de celular, el costo de minutos, entre otros.
6. System.out.println(data): Esta línea imprime en la consola la consulta SQL que se generó. Puede ser útil para fines de depuración para verificar que la consulta se haya formado correctamente.
7. statement.executeUpdate(data): Aquí se ejecuta la consulta SQL generada utilizando el objeto `Statement` creado previamente. La consulta realiza la inserción de datos en la base de datos.
8. obtenerConexion().close(): Se cierra la conexión con la base de datos.
9. catch (SQLException e): En caso de que ocurra una excepción de tipo `SQLException`, se captura y se maneja en este bloque. Se imprime un mensaje de excepción en la consola, que podría proporcionar información sobre el error que ocurrió.

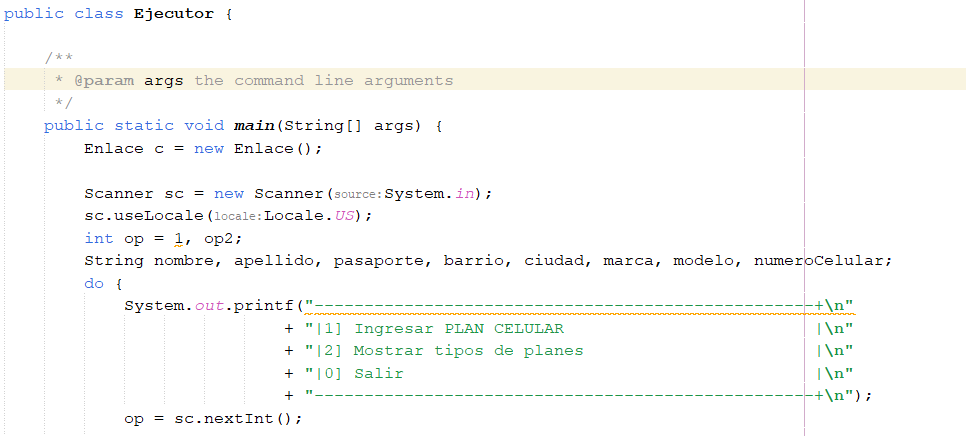
En resumen, este código realiza una inserción de datos en una base de datos utilizando una consulta SQL generada dinámicamente a partir de los valores de un objeto `PlanPostPagoMinutosMegasEconomico`.



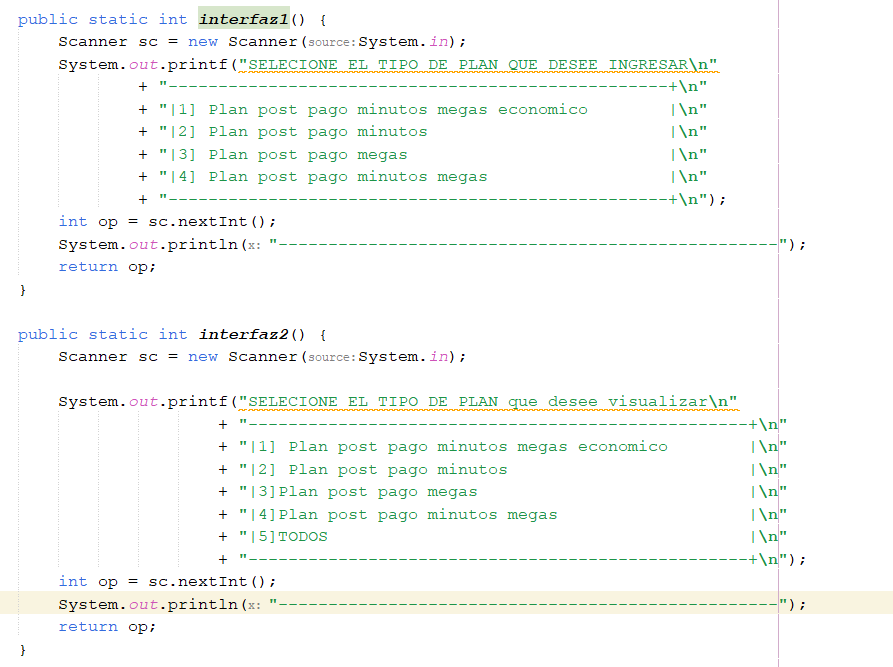
Este es el código mediante el cual obtenemos los datos y los uso para obtener los pagos, a esta información la almacena en un array list, este método también se usa en cada uno de los planes.

Por lo que al igual que en el caso anterior lo explicare a detalle:

1. Se crea una nueva instancia de `ArrayList<PlanPostPagoMinutos>` llamada `lista` que almacenará los objetos de tipo `PlanPostPagoMinutos`.
2. Se inicia un bloque `try` que maneja posibles excepciones de SQL.
3. Se llama al método `establecerConexion()` para establecer la conexión con la base de datos.
4. Se crea un objeto `Statement` a partir de la conexión obtenida mediante `obtenerConexion().createStatement()`.
5. Se define la consulta SQL en la cadena `data` para seleccionar todos los datos de la tabla `planPostPagoMinutos`.
6. Se ejecuta la consulta SQL utilizando el objeto `statement`, y se obtiene un objeto `ResultSet` llamado `rs` que contiene los resultados de la consulta.
7. A través de un bucle `while`, se recorren los resultados del conjunto de resultados (`rs`).
8. Se crean objetos `Propietario` con los datos obtenidos de la tabla y se almacenan en un objeto `p`.
9. Se crea un objeto `PlanPostPagoMinutos` con los datos obtenidos de la tabla y el objeto `p`. Se calcula el pago mensual llamando al método `calcularPagoMensual()`.
10. El objeto `PlanPostPagoMinutos` se agrega a la lista `lista`.
11. Se cierra la conexión con la base de datos.
12. En caso de una excepción de SQL, se imprime un mensaje de error junto con la información de la excepción.
13. Finalmente, se retorna la lista `lista` que contiene los objetos `PlanPostPagoMinutos` obtenidos de la base de datos.



Este es el ejecutor el cual funciona mediante un switch, este funciona con las variables op y op2, cada una llama a diferentes switch, op inicia igualada a uno y es la que da paso al menú principal, mientras que op2 que dependiendo del caso trenda un valor que llamara a menu1 y a menu2 respectivamente.

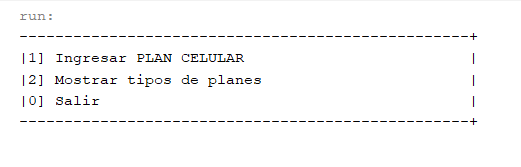


En esta parte del ejecutor se crean dos métodos, para mostrar dos diferentes menús con diferentes opciones, estos son los métodos:

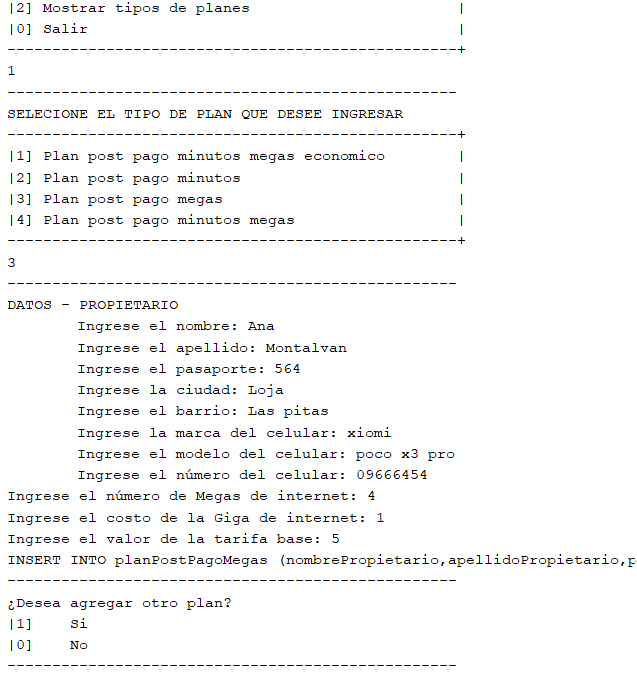
* Método interfaz1(): Este método muestra un menú donde el usuario puede seleccionar el tipo de plan que desea ingresar. Retorna la opción seleccionada.
* Método interfaz2(): Similar al método anterior, muestra un menú para que el usuario seleccione qué tipo de planes desea visualizar. Retorna la opción seleccionada.

## Resultados

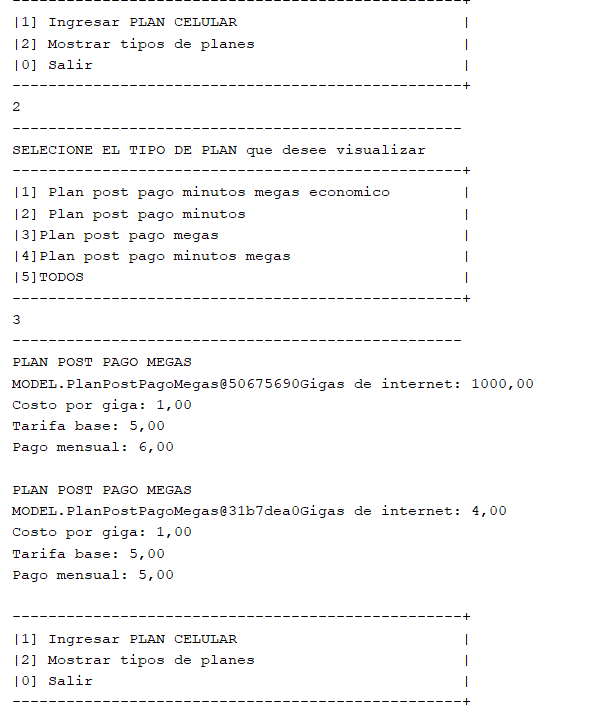
* Este es el primer menú.



* Luego dependiendo de la opción el programa te pedirá los datos necesarios.



* Aquí se muestra la segunda opción la cual presenta los planes guardados elegidos.



* Este finalmente es lo que sale cuando eliges la opción salir.

