**INGENIERÍA EN CIENCIAS DE LA COMPUTACIÓN**

**Informe Final: Desarrollo del Sistema RaccoonMovil**

**Programación Orientada a Objetos**

Est. Pablo Mateo Valdivieso Carrasco

Ing. Wayner Xavier Bustamante Granda

LOJA

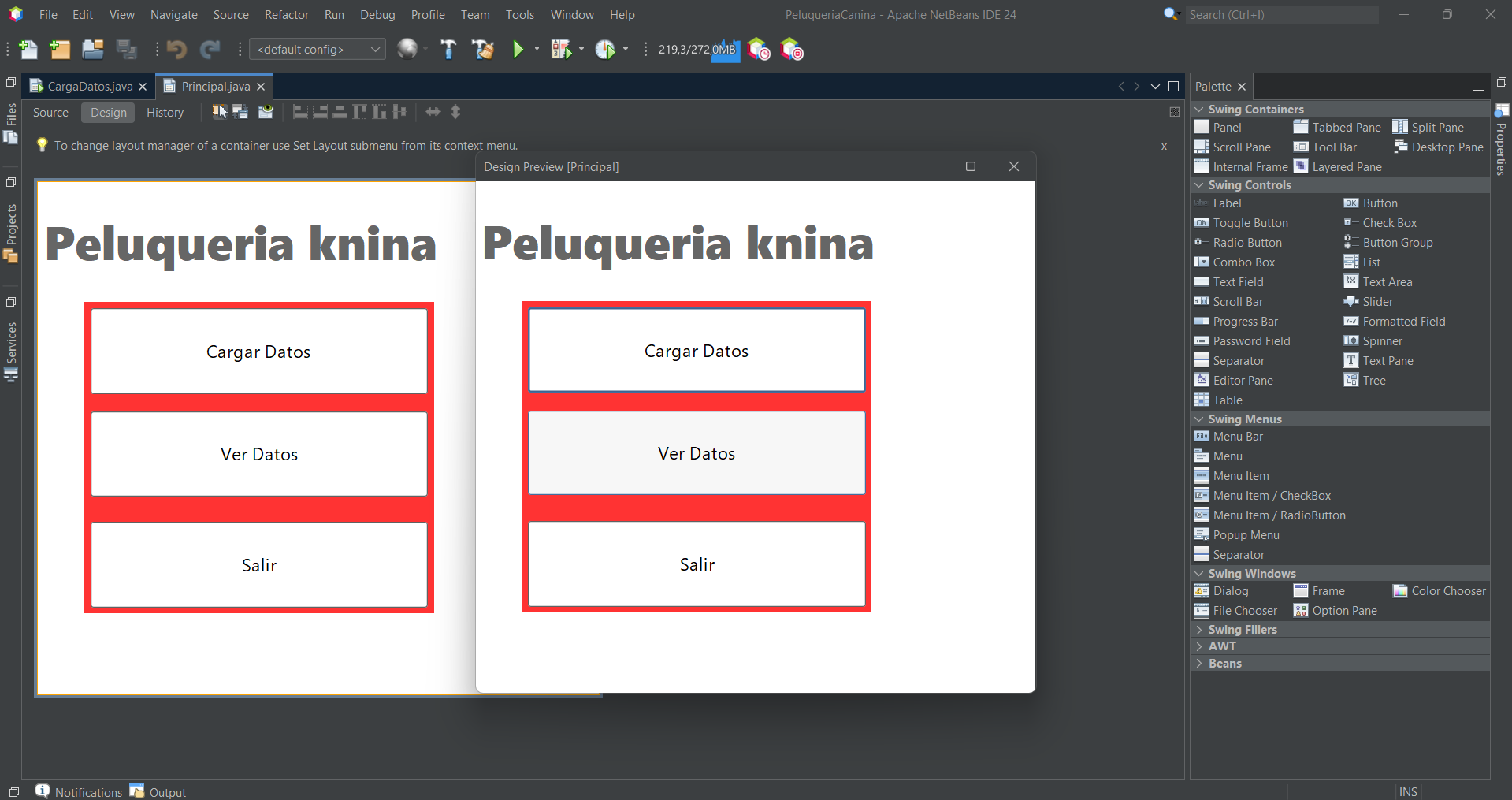
2025

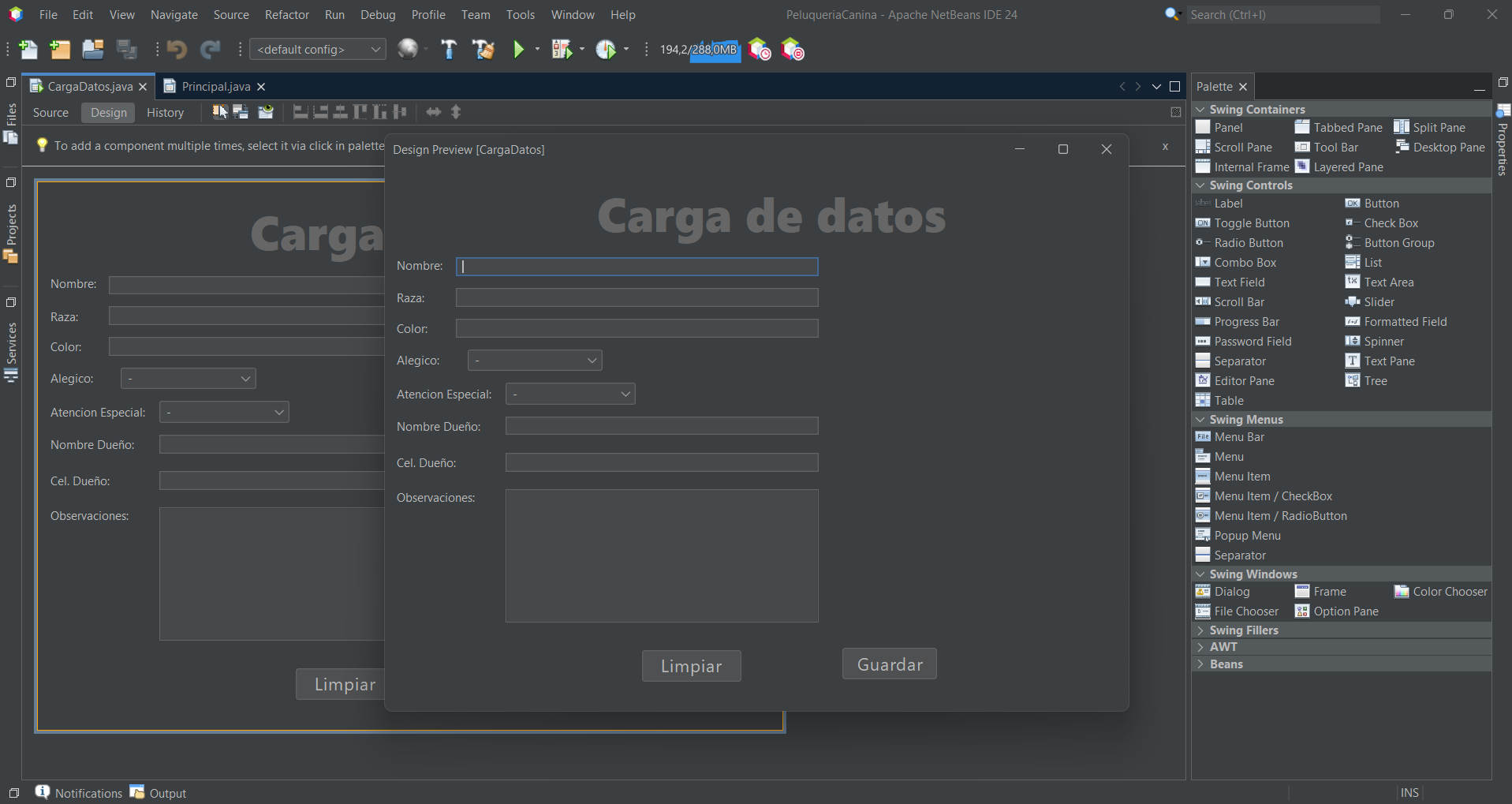
**Análisis:**

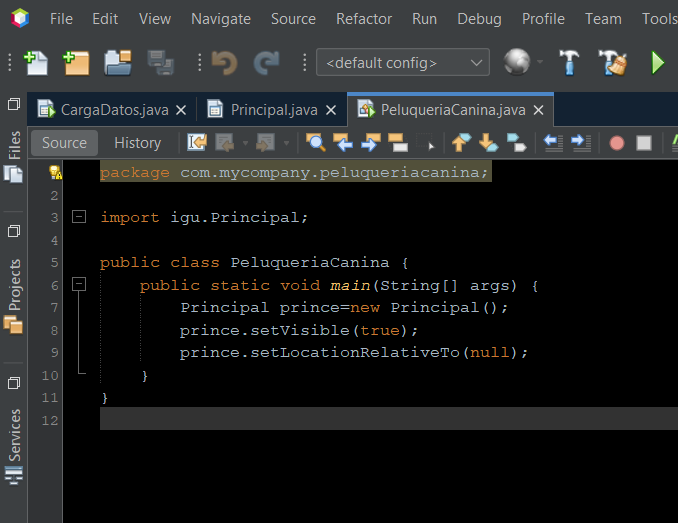
El problema planteado consistía en desarrollar una aplicación destinada al registro y facturación de planes telefónicos estudiantiles denominada "Mov-UTPL", sin embargo, decidí cambiar el nombre del proyecto renombrándolo así "RaccoonMovil". Aunque reconozco que la propuesta inicial tenía un enfoque práctico y muy ligado a la realidad universitaria, al inicio tuve ciertas dudas respecto a la motivación del proyecto, no por el tema en sí mismo, sino porque buscaba desarrollar algo más personal, algo que sintiera realmente mío.

Con esto en mente, opté por comenzar investigando cuál sería la mejor opción para realizar el programa. Conversando con algunos amigos de ciclos superiores, expertos en el área, ellos sugirieron emplear JFrame para la interfaz gráfica y MySQL para la base de datos, al ser opciones eficientes y sencillas para un proyecto de esta naturaleza. Convencido por sus recomendaciones, decidí adoptar esas herramientas.

Al profundizar más en mi investigación, encontré un canal de YouTube llamado “TodoCode” especializado en tutoriales prácticos sobre Programación Orientada a Objetos (POO), que incluía ejemplos de aplicaciones completas. Tomé como base uno de estos tutoriales para desarrollar una aplicación experimental denominada "Peluquería Canina" (a continuación se presentan capturas del programa como evidencia del proceso inicial de aprendizaje):



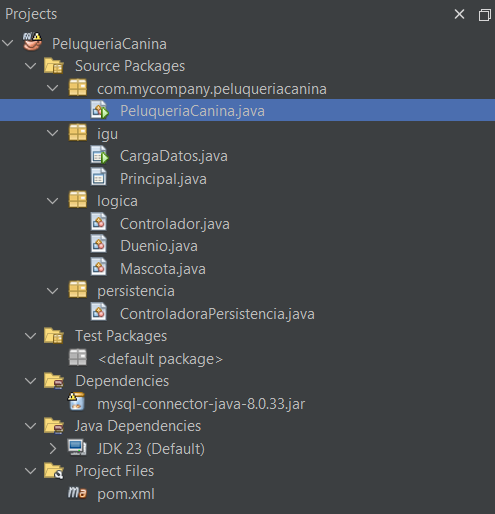




Gracias a esta aplicación inicial, logré entender a fondo los fundamentos del manejo gráfico en Java, específicamente en JFrame. Comprendí cómo usar adecuadamente botones, etiquetas (labels), áreas de texto y otros componentes visuales. En particular, seguí los pasos detallados en el video tutorial titulado "🔴 ¿Cómo crear una APP Java DESDE CERO? 👨🏻‍💻👩🏻‍💻 ¡Con CRUD con JPA! - Clase 100% PRÁCTICA (19-09)", el cual indicaba claramente la importancia de iniciar por la interfaz gráfica de usuario (GUI).

Este proyecto inicial lo estructuré en tres capas claramente definidas:

* **Capa gráfica (IGU)**: encargada de manejar la interacción directa con el usuario mediante interfaces como CargarDatos y Principal las gestionan la entrada y visualización de información.
* **Capa lógica (Logica)**: donde definí clases fundamentales como Controlador, Duenio y Mascota. Esta capa se encarga de manejar las operaciones centrales del sistema.
* **Capa de persistencia (Controller)**: implementada mediante la clase ControladoraPersistencia, que administra el almacenamiento y recuperación de datos desde/hacia la base de datos MySQL Workbench.



Siguiendo estos lineamientos, diseñé inicialmente la interfaz gráfica, configurando posteriormente cada botón para asignarles funciones específicas: uno para cerrar el programa, otro para mostrar los datos almacenados y otro para insertar nuevos registros. Cada botón redirigía a una interfaz nueva o cerraba adecuadamente la aplicación, lo cual me permitió adquirir la habilidad de gestionar múltiples ventanas y eventos en una aplicación gráfica.

Otro aspecto fundamental que aprendí fue la manera adecuada de capturar datos del usuario a través de formularios visuales, descartando así el uso tradicional de la clase Scanner. Este enfoque simplificó notablemente la interacción usuario-sistema, facilitando una experiencia mucho más intuitiva y amigable.

Finalmente, con estos conocimientos previos y ya familiarizado con la creación y manejo de interfaces gráficas y bases de datos básicas, procedí a abordar el desarrollo completo del proyecto "RaccoonMovil". Este primer análisis y aprendizaje resultó crucial para sentar bases sólidas que garantizarían el éxito y eficiencia del sistema final.

**Diseño:**

El diseño UML es una herramienta fundamental en el desarrollo del sistema RaccoonMovil, permitiendo una representación visual clara y levemente ordenada de la estructura del software, facilitando así su comprensión, implementación y mantenimiento.

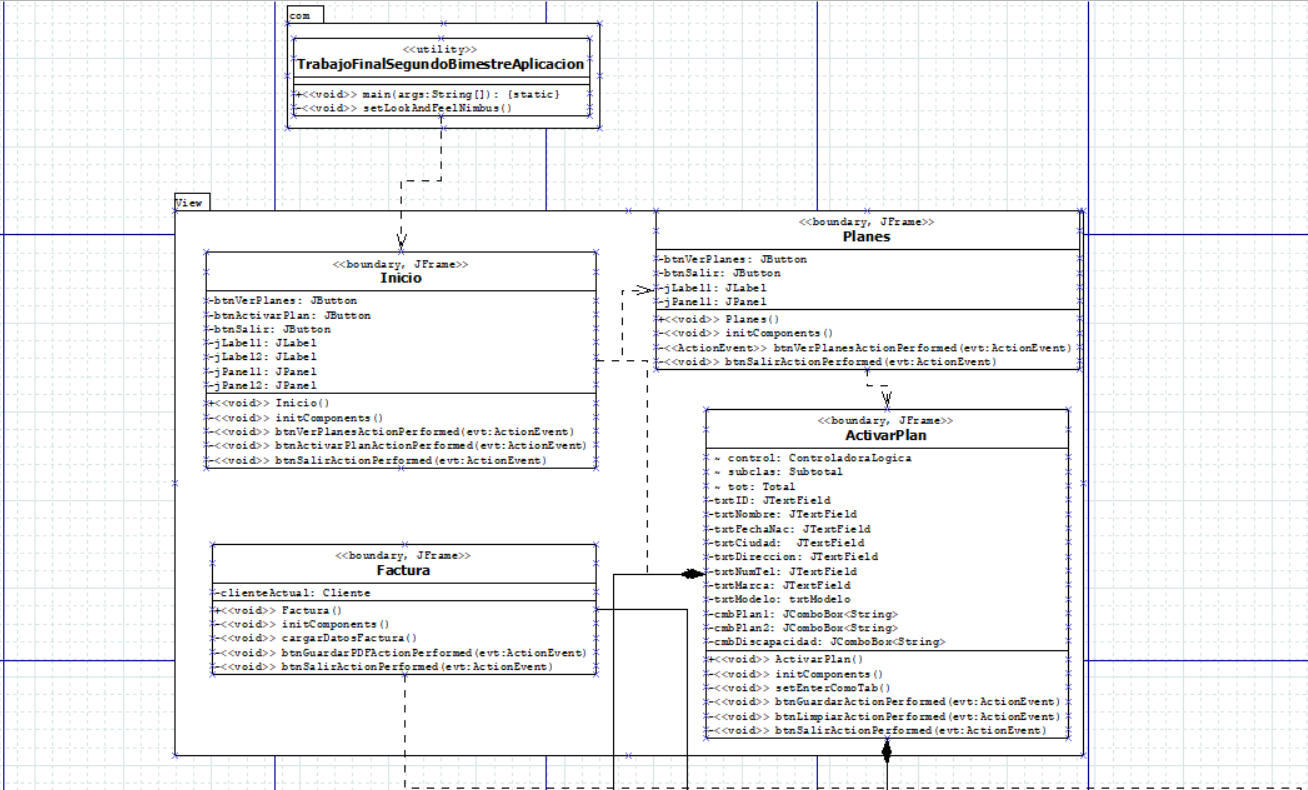
El proyecto RaccoonMovil está organizado en cinco paquetes claramente definidos:

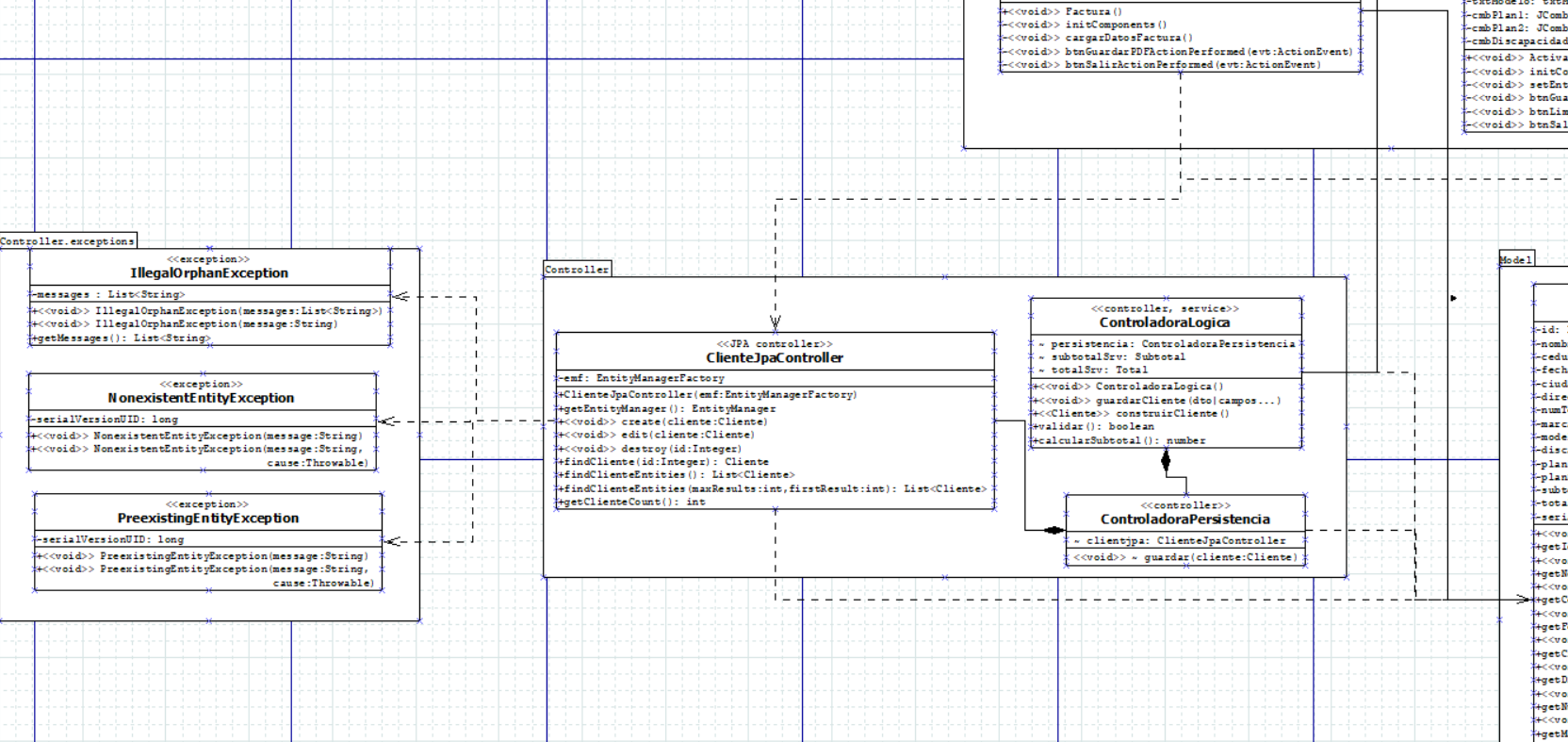
* **View**: contiene las interfaces gráficas de usuario que permiten la interacción directa, como Inicio, ActivarPlan, Factura y Planes. Estas interfaces gestionan la entrada de datos por parte del usuario y la visualización de resultados.
* **Controller**: contiene las clases encargadas de la lógica del negocio y la interacción con la base de datos, tales como ClienteJpaController, ControladoraLogica y ControladoraPersistencia. Estas clases realizan las operaciones CRUD necesarias para el manejo efectivo de los datos.
* **Controller.exceptions**: compuesto por las excepciones personalizadas del sistema como IllegalOrphanException, NonexistentEntityException y PreexistingEntityException, utilizadas para manejar situaciones especiales durante las operaciones CRUD.
* **Model**: alberga todas las clases fundamentales del negocio, incluyendo entidades como Cliente, los diferentes tipos de planes (Planprecio, Racconomic, RaccoonStar, InternationalRaccoon, OnlineRaccoon) y los descuentos específicos (DescuentoMayorEdad, DescuentoDiscapacidad, DescuentoAmbas, DescuentoNinguno). Aquí se define la estructura básica y comportamiento central del sistema.
* **Com**: paquete auxiliar que contiene la clase principal de la aplicación TrabajoFinalSegundoBimestreAplicacion, encargada de iniciar la aplicación y gestionar aspectos globales del sistema.

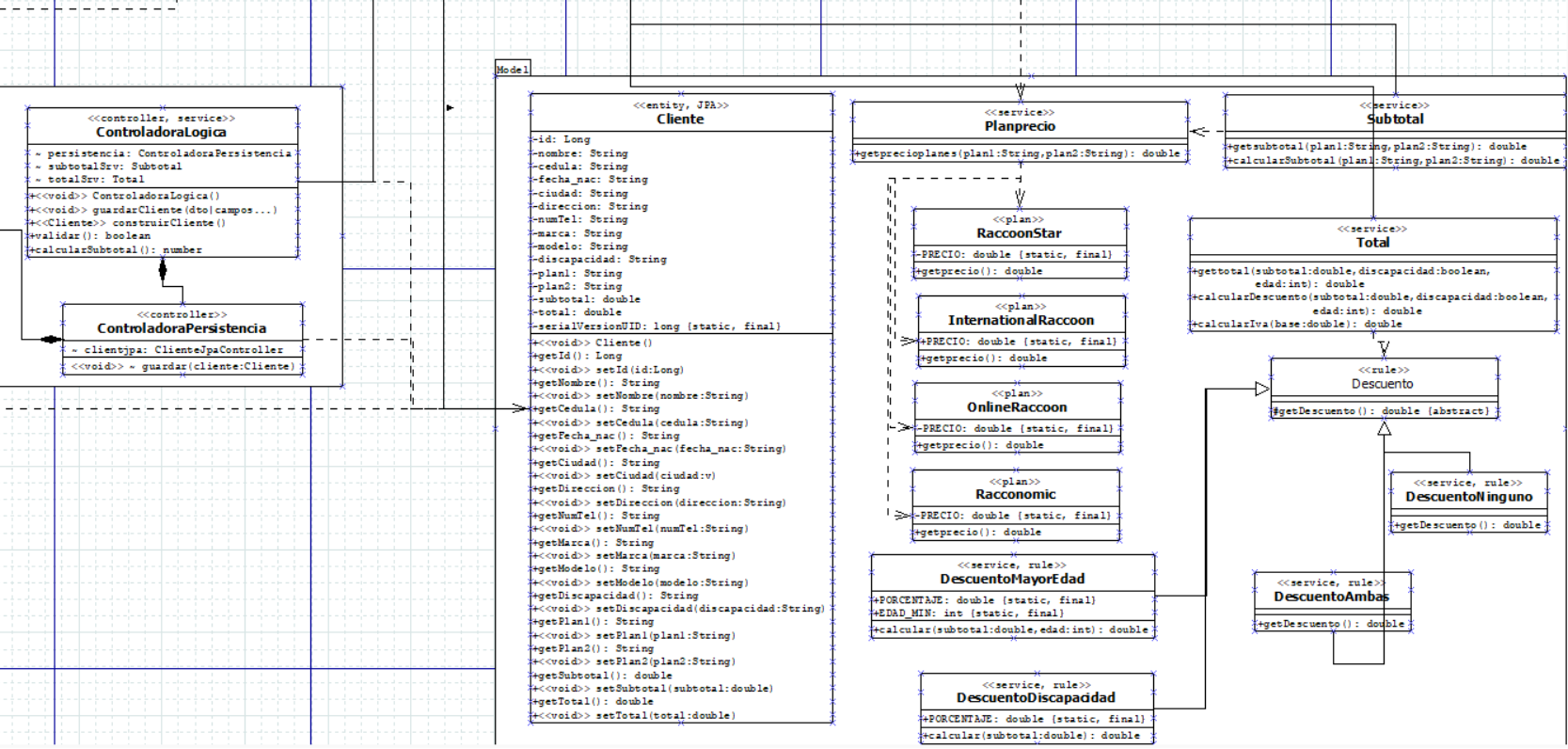
Relaciones entre las clases:

* TrabajoFinalSegundoBimestreAplicacion crea (Dependency) la interfaz Inicio.
* Inicio abre (Dependency) las interfaces Planes y ActivarPlan.
* Planes también abre (Dependency) la interfaz ActivarPlan.
* ActivarPlan contiene una composición con ControladoraLogica, Subtotal y Total.
* Factura está asociada a la entidad Cliente y utiliza (Dependency) las clases Planprecio y ClienteJpaController.
* ControladoraLogica contiene una composición con ControladoraPersistencia y utiliza (Dependency) la clase Cliente.
* ControladoraPersistencia contiene una composición con ClienteJpaController y utiliza (Dependency) la clase Cliente.
* ClienteJpaController realiza operaciones CRUD (Dependency) sobre la entidad Cliente y maneja excepciones (Dependency) definidas en Controller.exceptions.
* Subtotal utiliza (Dependency) Planprecio, la cual a su vez interactúa con diferentes tipos de planes.
* Total utiliza un patrón estrategia (Dependency) para manejar los descuentos específicos mediante la clase abstracta Descuento, de la cual heredan diferentes implementaciones (DescuentoDiscapacidad, DescuentoMayorEdad, DescuentoAmbas, DescuentoNinguno).

A continuación se muestran los diagramas UML realizados en DIA que ilustran gráficamente todas estas relaciones y estructuras mencionadas:







**Codificación:**

A continuación se presentan los fragmentos de código más representativos del proyecto **RaccoonMovil**, acompañados de comentarios explicativos. Además, se reconoce la ayuda recibida de un compañero de cuarto ciclo y de la IA ChatGPT para el diseño del logo.

**1. Clase principal (TrabajoFinalSegundoBimestreAplicacion.java):**

Clase principal que inicia la aplicación. Se mantiene el main corto por buenas prácticas:

package com.mycompany.trabajofinalsegundobimestreaplicacion;

import View.Inicio;

public class TrabajoFinalSegundoBimestreAplicacion {

public static void main(String[] args) {

Inicio start=new Inicio();

start.setVisible(true);

start.setLocationRelativeTo(null);

}

}

**2. Controladora lógica (ControladoraLogica.java):**

Encapsula la lógica de negocio de RaccoonMovil. Recibí ayuda de un compañero de 4to ciclo para optimizar estos métodos.

public class ControladoraLogica {

private ControladoraPersistencia persistencia = new ControladoraPersistencia();

/\*\*

\* Construye y guarda un nuevo cliente con todos sus atributos.

\*/

public void guardarCliente(String nombre, String cedula, String ciudad, String direccion,

String fecha\_nac, String marca, String modelo, String numTel,

String discapacidad, String plan1, String plan2,

double subtotal, double total) {

Cliente cliente = new Cliente();

// Asignación de atributos

cliente.setNombre(nombre);

cliente.setCedula(cedula);

cliente.setCiudad(ciudad);

cliente.setDireccion(direccion);

cliente.setFecha\_nac(fecha\_nac);

cliente.setMarca(marca);

cliente.setModelo(modelo);

cliente.setNumTel(numTel);

cliente.setDiscapacidad(discapacidad);

cliente.setPlan1(plan1);

cliente.setPlan2(plan2);

cliente.setSubtotal(subtotal);

cliente.setTotal(total);

// Delegar persistencia

persistencia.guardar(cliente);

}

}

**3. Manejo de Excepciones y CRUD (ClienteJpaController.java):**

package Controller;

import Controller.exceptions.NonexistentEntityException;

import Model.Cliente;

import javax.persistence.\*;

import java.io.Serializable;

import java.util.List;

public class ClienteJpaController implements Serializable {

private EntityManagerFactory emf = Persistence.createEntityManagerFactory("RaccoonMovilPU");

/\*\*

\* Crea un nuevo registro de Cliente en la base de datos.

\*/

public void create(Cliente cliente) {

EntityManager em = emf.createEntityManager();

try {

em.getTransaction().begin();

em.persist(cliente);

em.getTransaction().commit();

} finally {

em.close();

}

}

/\*\*

\* Edita un Cliente existente, lanza excepción si no existe.

\*/

public void edit(Cliente cliente) throws NonexistentEntityException, Exception {

EntityManager em = emf.createEntityManager();

try {

em.getTransaction().begin();

em.merge(cliente);

em.getTransaction().commit();

} catch (Exception ex) {

Long id = cliente.getId();

if (findCliente(id) == null) {

throw new NonexistentEntityException("El cliente con id " + id + " no existe.");

}

throw ex;

} finally {

em.close();

}

}

// Métodos adicionales: destroy, findCliente, findClienteEntities, getClienteCount

**4. Implementación de Estrategia de Descuento (Total.java y DescuentoMayorEdad.java):**

// Total.java

package Model;

/\*\*

\* Calcula el total aplicado el IVA.

\*/

public class Total {

private final double IVA\_FACTOR = 1.15;

private double subtotal;

public Total(double subtotal) {

this.subtotal = subtotal;

}

public double calcularTotal() {

return subtotal \* IVA\_FACTOR;

}

}

// DescuentoMayorEdad.java

package Model;

/\*\*

\* Descuento para clientes de 65 años o más.

\*/

public class DescuentoMayorEdad extends Descuento {

@Override

public double getDescuento() {

return 0.50; // 50% de descuento

}

}

**5. Persistencia simplificada (ControladoraPersistencia.java):**

package Controller;

import Model.Cliente;

/\*\*

\* Administra la capa de persistencia delegando en JPA.

\*/

public class ControladoraPersistencia {

private ClienteJpaController clientJpa = new ClienteJpaController();

public void guardar(Cliente cliente) {

clientJpa.create(cliente);

}

}

**6. Ejemplo de Evento en la Interfaz (ActivarPlan.java):**

// Fragmento dentro de ActivarPlan.java (manejador del botón Guardar)

private void btnGuardarActionPerformed(java.awt.event.ActionEvent evt) {

// Validación de campos obligatorios

if (ciudad.isEmpty() || id.isEmpty() || nombre.isEmpty()) {

JOptionPane.showMessageDialog(null, "Faltan datos obligatorios.");

return;

}

// Cálculo de subtotal y total

Subtotal sub = new Subtotal();

double subtotal = sub.calcularSubtotal(plan1, plan2, edad, discapacidad);

Total tot = new Total(subtotal);

double total = tot.calcularTotal();

// Guardar cliente

control.guardar(ciudad, direccion, fecha\_nac, id, marca, modelo,

nombre, numTel, discapacidad, plan1, plan2, subtotal, total);

// Mostrar factura

new Factura().setVisible(true);

this.dispose();

}

**Uso de IA (ChatGPT) en el proyecto:**

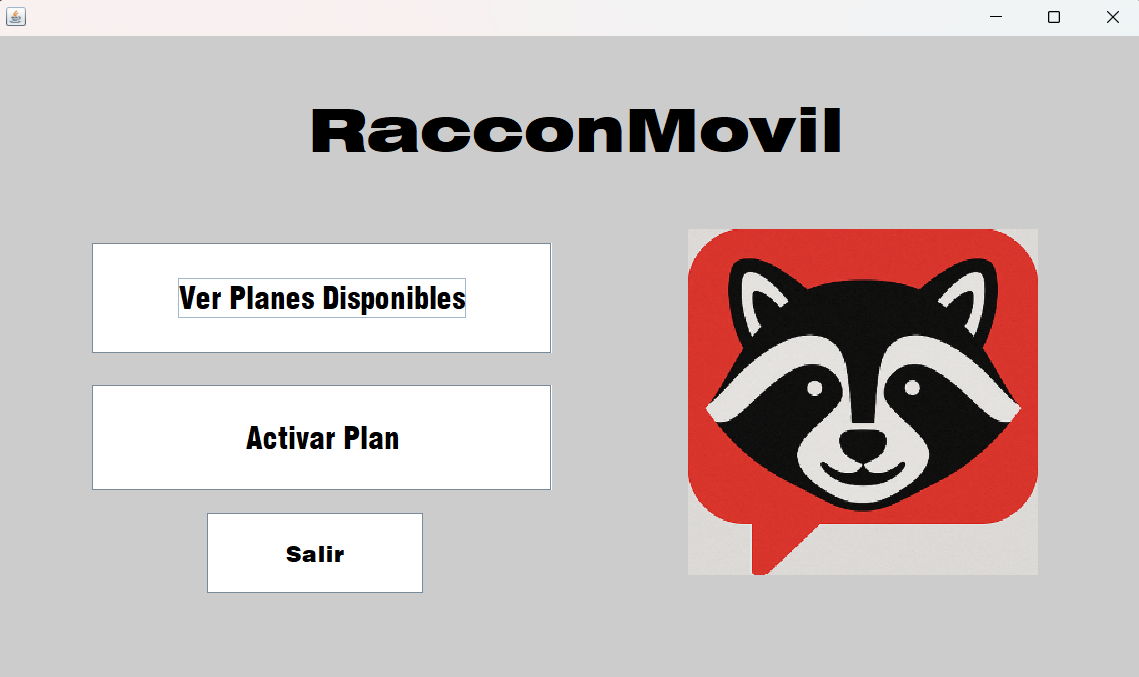
Para diseñar el logo de la operadora —caracterizada por un mapache— utilicé la ayuda de ChatGPT. La IA me proporcionó ideas de composición, paleta de colores y estilo, logrando un logo creativo y acorde con la identidad de RaccoonMovil.

****

**Resultados:**

A continuación se muestran ejemplos de pantallas y outputs obtenidos al ejecutar RaccoonMovil:

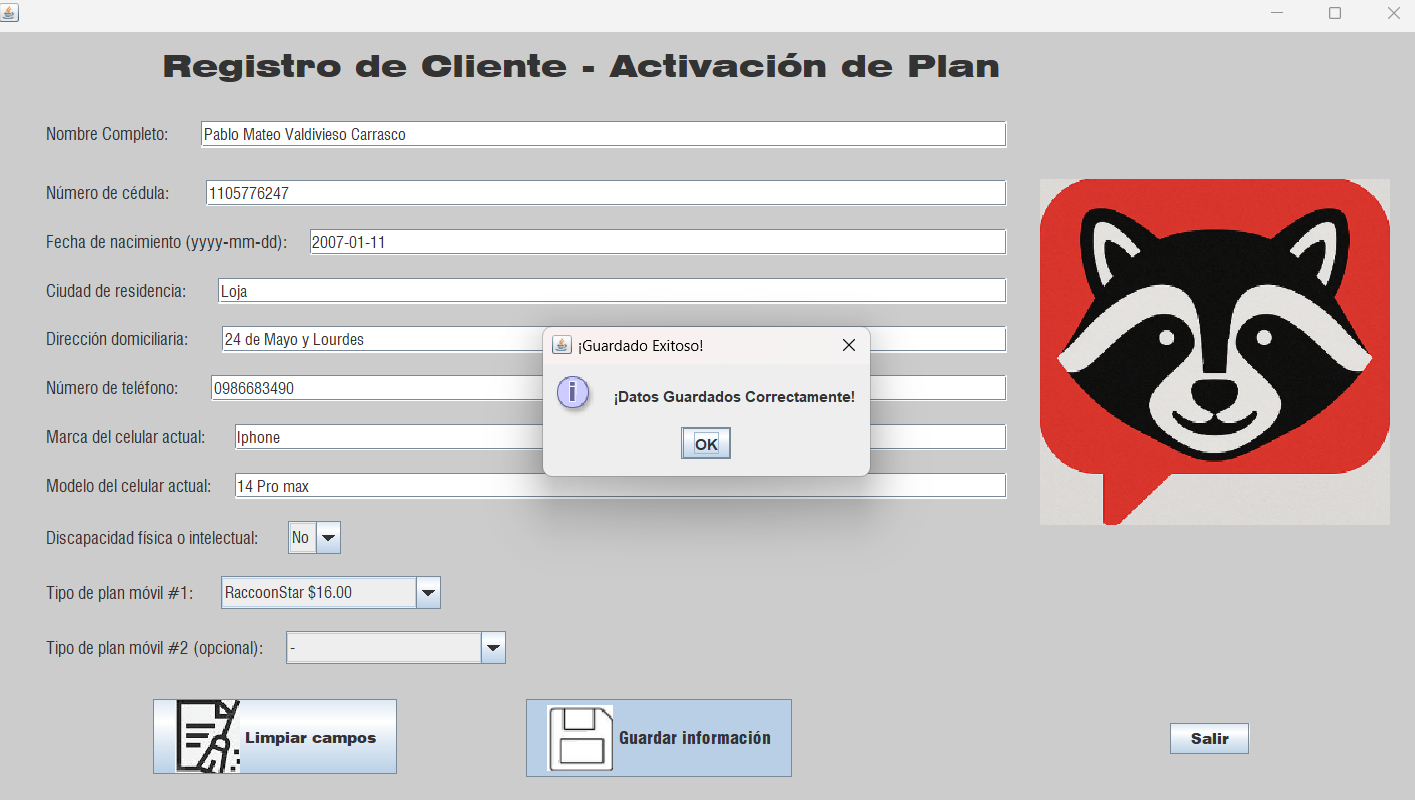
1. **Pantalla de inicio:** muestra las opciones principales (Ingresar cliente, Activar plan, Ver facturas).



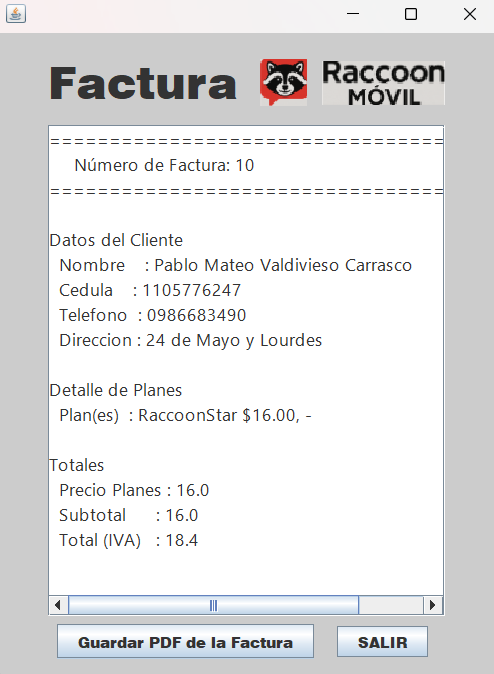
1. **Formulario de Activar Plan:** donde el usuario ingresa datos y selecciona planes.

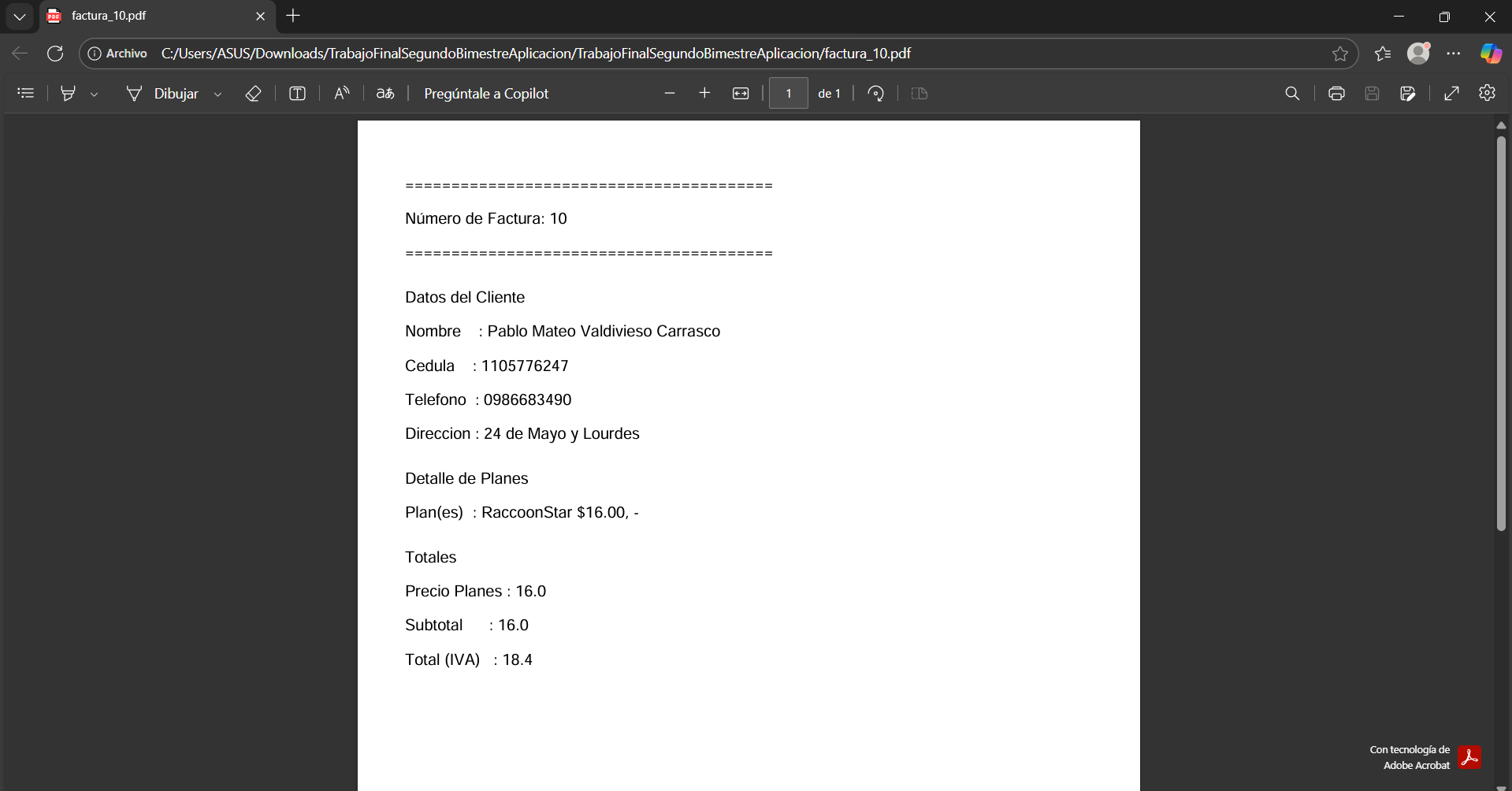


1. **Mensaje de confirmación:** diálogo emergente indicando "Datos Guardados Correctamente".

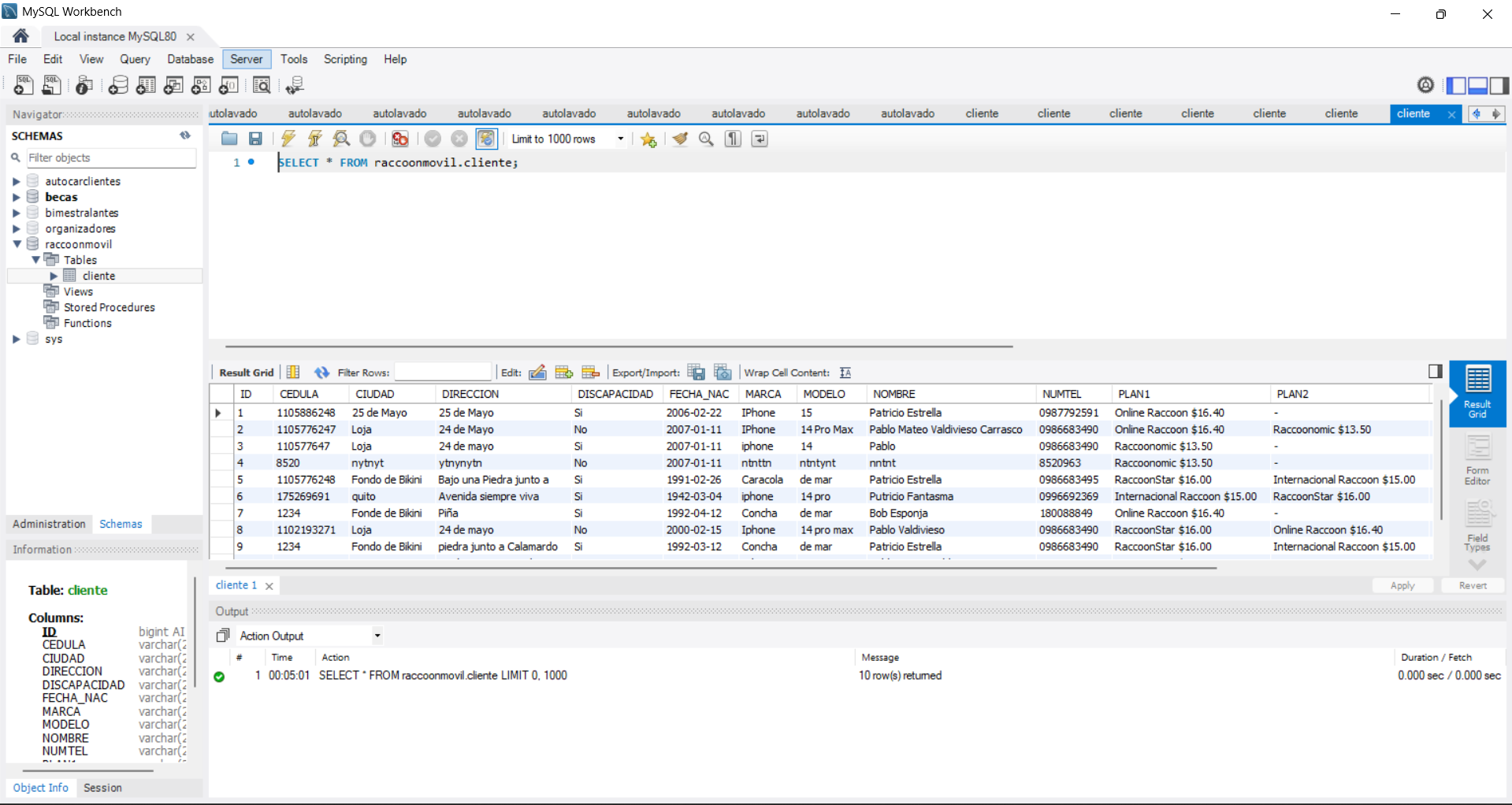


1. **Factura generada:** ventana que presenta los detalles del cliente, subtotal, descuento aplicado y total.





1. **Listado de clientes en la base de datos:** salida en consola o tabla mostrando registros almacenados.



Estas capturas evidencian el correcto flujo de ingreso de datos y generación de facturas, validaciones de campos y aplicación de descuentos en RaccoonMovil.

**Bibliografía**

*ChatGPT*. (s/f). Chatgpt.com. Recuperado el 30 de julio de 2025, de <https://chatgpt.com/>

*¿Cómo habilitar la tecla “Enter” en un formulario de Login en Java Swing?* (s/f). Stack Overflow en español. Recuperado el 30 de julio de 2025, de <https://es.stackoverflow.com/questions/175787/c%C3%B3mo-habilitar-la-tecla-enter-en-un-formulario-de-login-en-java-swing>

de Paula, L. (2022, septiembre 20). *🔴 ¿Cómo crear una APP Java DESDE CERO? 👨🏻‍💻👩🏻‍💻 ¡Con CRUD con JPA! - Clase 100% PRÁCTICA (19-09)*. Youtube. <https://www.youtube.com/watch?v=Gd4QQtQz5DE&t=2651s>

*Postgres sandbox*. (s/f). Database.Build. Recuperado el 30 de julio de 2025, de <https://database.build/db/db6f2f35ouj9lokm>