UNIVERSIDAD MAYOR DE SAN ANDRES

CARRERA DE INFORMATICA

PROGRAMACION II

INF-121

Autores : **Romel Robhino Quispe Cayo**

Limachi Laura José Fernando

Herrera Palomino Deyna Gabriela

Tarquino Tuco Rene Gonzalo

Muñoz Zarzure Josue

Materia: **Programación II**

Docente: **Rosalia Lopez Montalvo**

Fecha: **24 de julio de 2025**

LA PAZ - BOLIVIA

2025

1.Resumen del proyecto

Introducción:
Este proyecto consiste en el diseño e implementación de un sistema de gestión de parqueo orientado a objetos, con una interfaz de usuario gráfica básica. El sistema busca automatizar y optimizar las operaciones de registro de vehículos, asignación de espacios, y gestión de la información de clientes y tarifas.

- **¿Qué problema soluciona?**
  - Este sistema aborda la problemática de la gestión manual o ineficiente de parqueos, proporcionando una herramienta para registrar de forma digital las entradas y salidas de vehículos, asignarles espacios, asociarlos a clientes, y calcular automáticamente las tarifas. Esto reduce errores, mejora el control y agiliza las operaciones diarias del parqueo.
- **¿Qué funcionalidades tiene?**
  - Registro de información de Clientes (nombre).
  - Registro de información de Vehículos (placa, marca, color, tipo, propietario).
  - Asignación de espacio a un vehículo.
  - Registro de hora de ingreso y salida de vehículos.
  - Cálculo de la duración de la estadía y tarifa a pagar.
  - Verificación básica de disponibilidad de espacios.
  - Persistencia de datos de vehículos en una base de datos MySQL.
  - Interfaz gráfica de usuario para la interacción con el sistema.
- Tecnologías usadas:
  - Lenguaje de Programación: Java
  - IDE: Eclipse, NetBeans
  - Interfaz Gráfica: Java Swing
  - Persistencia: MySQL (a través de JDBC)
  - Librerías: java.time.* para manejo de fechas y horas.

2. Objetivos

Diseñar e implementar un sistema de gestión de parqueo utilizando los principios de la programación orientada a objetos, estructuras de datos y, cuando sea aplicable, patrones de diseño.

2.1. Objetivos Específicos

- Aplicar conceptos de encapsulamiento, herencia (implícita en la gestión del tipo de Vehiculo), y manejo de asociaciones entre objetos (Cliente y Vehiculo).
- Utilizar estructuras de datos genéricas de Java para la gestión de colecciones de objetos (ej. List<Vehiculo> en Registro).
- Implementar la persistencia de los datos de vehículos utilizando una base de datos relacional (MySQL) a través de JDBC.
- Desarrollar una interfaz de usuario básica con Java Swing para facilitar la interacción con el sistema.
- Simular una interacción funcional que permita el registro de clientes y vehículos, y la visualización de su información.

3. Análisis del Problema

3.1. Descripción del Contexto

Los parqueos modernos requieren sistemas eficientes para manejar el alto volumen de vehículos y optimizar el uso del espacio. La gestión manual es propensa a errores, ineficiente en el cálculo de tiempos y tarifas, y carece de un registro centralizado y accesible. Un sistema automatizado se vuelve indispensable para ofrecer un servicio ágil y confiable, además de proporcionar datos para la toma de decisiones.

3.2. Requisitos Funcionales (mínimo 5)

1. El sistema debe permitir registrar la información de un nuevo cliente (nombre).
2. El sistema debe permitir registrar la entrada de un vehículo, capturando su placa, marca, color, tipo y asociándolo a un cliente existente, además de asignarle un espacio y registrar la hora de ingreso.
3. El sistema debe permitir registrar la salida de un vehículo, calcular la duración de la estadía y el monto a pagar según la tarifa.
4. El sistema debe verificar si un espacio de parqueo está disponible antes de asignarlo.
5. El sistema debe persistir los datos de los vehículos y sus propietarios en una base de datos para su recuperación futura.
6. El sistema debe calcular la tarifa basándose en las horas completas de permanencia.
7. El sistema debe permitir consultar los datos de un vehículo y su propietario.

3.3. Requisitos No Funcionales (mínimo 3)

1. **Usabilidad:** La interfaz de usuario debe ser intuitiva y fácil de usar para el personal del parqueo.
2. **Fiabilidad:** Los datos registrados en la base de datos deben ser consistentes y recuperables.
3. **Rendimiento:** Las operaciones de registro y consulta de datos deben ser rápidas y eficientes para no ralentizar el flujo de vehículos.
4. **Conectividad:** El sistema debe establecer y mantener una conexión estable con la base de datos MySQL.

3.4. Casos de Uso (breve descripción por funcionalidad)

- **Registrar Cliente:** El usuario ingresa el nombre de un nuevo cliente a través de un formulario y el sistema lo guarda.
- **Registrar Vehículo y Entrada:** El usuario ingresa los datos de un vehículo (placa, marca, color, tipo) y selecciona un cliente propietario. El sistema asigna un espacio, registra la hora de ingreso y guarda los datos en la base de datos.
- **Registrar Salida de Vehículo:** El usuario busca un vehículo por su placa, el sistema recupera sus datos, calcula la duración y la tarifa, y libera el espacio.
- **Consultar Datos de Vehículo:** El usuario ingresa una placa y el sistema muestra los detalles del vehículo, incluyendo su propietario y la ubicación asignada.

4. Diseño del Sistema

4.1. Clases y Jerarquías

El diseño del sistema se centra en la interacción de entidades clave para la gestión del parqueo. Aunque el diseño original del diagrama UML sugería una jerarquía de Vehiculo con subclases, la implementación

actual utiliza la clase Vehiculo de forma concreta, diferenciando los tipos a través de un atributo String. La interfaz gráfica (Swing) introduce clases de formulario (ClienteForm, VehiculoForm) para la interacción con el usuario.

4. Diseño del Sistema

4.1. Clases y Jerarquías

El diseño del sistema se centra en la interacción de entidades clave para la gestión del parqueo. Aunque el diseño original del diagrama UML sugería una jerarquía de Vehiculo con subclases, la implementación actual utiliza la clase Vehiculo de forma concreta, diferenciando los tipos a través de un atributo String. La interfaz gráfica (Swing) introduce clases de formulario (ClienteForm, VehiculoForm) para la interacción con el usuario.

Diagrama UML: (Se hace referencia al diagrama que se adjuntó previamente, mostrando las clases principales: Parqueo, Registro, ControlIngreso, Tarifa, UbicacionParqueo, Vehiculo, Cliente, Conexion. Adicionalmente, se reconocen las clases de interfaz ClienteForm y VehiculoForm.)

Tabla de Clases:

CLASE	ATRIBUTOS PRINCIPALES	METODOS
Cliente	nombre, vehiculo	getNombre(), getVehiculo(), setVehiculo(), obtenerDatos()
ClienteForm	campoNombre, botonGuardar, clienteCreadoListener	setClienteCreadoListener(),ClienteForm(), guardarCliente()
Conexión	URL, USUARIO, CLAVE	conectar()
ControlIgreso	tarifa, registro, espacio	verificarDisponibilidad(), asignarUbicacion(), registrarEntrada(), registrarSalida()
Main		main()
Parqueo	id, espaciosOcupados, totalEspacios, barrera	estaDisponible(), ocuparEspacio(), liberarEspacio(), abrirBarrera(), cerrarBarrera()
Registro	ingresos<Vehiculo>, salidas<Vehiculo>	guardarIngreso(), guardarSalida(), buscarVehiculoPorPlaca()
Tarifa	tarifaPorHora	calcularTarifa(), getTarifaPorHora(
UbicacionParqueo	planta, carril, casilla, ocupado	estaDisponible(), asignarVehiculo(), liberar(), obtenerUbicacion()
Vehiculo	placa, marca, color, propietario, tipo, espacioAsignado, horaIngreso, horaSalida	asignarEspacio(), registrarIngreso(), registrarSalida(), obtenerDuracion(), obtenerDatos(), getPlaca(), getPropietario(), getHoraIngreso(), getHoraSalida(), setPropietario()
VehiculoForm	campoPlaca, campoMarca, campoColor, campoTipo, campoEspacio, botonGuardar, propietario	VehiculoForm(), guardarVehiculo()

4.2. Relaciones (Asociación, Agregación, Composición)

- **Asociación:**
  - Cliente está asociado con Vehiculo (un cliente puede tener un vehículo, y un vehículo tiene un propietario cliente). Esta es una relación bidireccional donde Cliente tiene una referencia a Vehiculo y Vehiculo tiene una referencia a Cliente.
  - ControlIngreso está asociado con Tarifa, Registro, y UbicacionParqueo para realizar sus operaciones.
- **Agregación:**
  - Registro agrega listas de Vehiculos (ingresos y salidas). Los vehículos pueden existir independientemente de los registros.
  - ControlIngreso agrega una UbicacionParqueo para gestionar su estado de disponibilidad.

Interacción entre Clases:

- Main inicia las ClienteForm y VehiculoForm, que son las interfaces de usuario.
- ClienteForm crea objetos Cliente.
- VehiculoForm crea objetos Vehiculo y los asocia con un Cliente existente. También maneja la persistencia de Vehiculo en la base de datos a través de Conexion.
- ControlIngreso coordina la lógica de negocio para entrada/salida de vehículos, utilizando Tarifa para cálculos, Registro para guardar eventos y UbicacionParqueo para gestionar los espacios físicos.
- Vehiculo registra sus propias horas de ingreso/salida y se asocia a un Cliente y a un espacioAsignado (String).

5. Desarrollo

5.1. Estructura General del Código

Todas las clases del proyecto se encuentran en un único paquete: parqueo_inteligente.newpackage. Esta organización facilita la visibilidad entre las clases pero, para proyectos más grandes, se recomienda una división en subpaquetes (modelo, servicio, persistencia, gui, etc.) para una mayor modularidad.

5.2. Uso de Genéricos

Se hace uso de la interfaz List y su implementación ArrayList para manejar colecciones de objetos Vehiculo dentro de la clase Registro, lo que proporciona seguridad de tipo en tiempo de compilación.

5.3. Métodos Sobrescritos

El método toString() es sobrescrito en la clase Cliente para proporcionar una representación de cadena útil del objeto. El método obtenerDatos() en Cliente y Vehiculo también cumple una función similar, mostrando información formateada.

5.4. Validaciones

Las validaciones básicas se implementan en los formularios de la interfaz gráfica (ClienteForm y VehiculoForm) para asegurar que los campos obligatorios no estén vacíos antes de intentar guardar la información.

5.5. Persistencia con Base de Datos (MySQL)

El sistema utiliza una base de datos MySQL para la persistencia de los datos de los vehículos. La conexión a la base de datos se maneja a través de la clase Conexion y las operaciones de inserción se realizan directamente en VehiculoForm usando JDBC.

5.6. Interfaz de Usuario Gráfica (Swing)

El proyecto incluye una interfaz de usuario gráfica desarrollada con Java Swing, lo que permite una interacción más amigable con el usuario en comparación con una aplicación de consola. Las clases ClienteForm y VehiculoForm son los componentes principales de esta interfaz, facilitando la entrada de datos.

6. Aplicación de Patrones de Diseño

Basado en el código proporcionado, se identifican las siguientes aplicaciones de patrones:

6.1. Patrón(es) Aplicados

Patrón	Clases Involucradas	Rol en el Sistema
Listener (Callback)	ClienteForm, VehiculoForm, ClienteCreadoListener	Permite la comunicación entre el formulario de cliente y el de vehículo.

6.2. Justificación

- **Listener (Callback):** Se utiliza una interfaz ClienteCreadoListener en ClienteForm para notificar a otras partes del sistema (en este caso, Main que luego lanza VehiculoForm) cuando un cliente ha sido creado y guardado exitosamente. Esto desacopla la creación del cliente de la acción subsiguiente (creación del vehículo), permitiendo que ClienteForm sea genérico y no necesite saber qué hacer después de guardar un cliente, solo que debe notificar a sus oyentes.

7. Persistencia de Datos

7.1. Descripción del Medio Usado (MySQL)

La persistencia de los datos del sistema se maneja a través de una base de datos relacional MySQL. Esta elección permite una gestión estructurada de los datos, integridad referencial y escalabilidad para futuras expansiones del sistema. Se utiliza JDBC (Java Database Connectivity) para interactuar con la base de datos.

7.2. Clases Encargadas de la Conexión y Operaciones

- **Conexion:** Esta clase es la encargada de establecer la conexión con la base de datos MySQL. Contiene los parámetros de conexión (URL, usuario, clave) y el método estático conectar() que devuelve un objeto Connection.
- **VehiculoForm:** En la implementación actual, la lógica de inserción de datos del vehículo directamente en la base de datos se encuentra dentro del método guardarVehiculo() de la clase VehiculoForm. Esto incluye la preparación de la sentencia SQL y su ejecución.

7.3. Ejemplo de Interacción con la Base de Datos

El siguiente fragmento de código muestra cómo VehiculoForm se conecta a la base de datos y guarda la información de un vehículo:

```java
package parqueo_inteligente.newpackage;

import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.SQLException;
import java.time.LocalTime;

public class VehiculoForm extends JFrame {
    private JTextField campoPlaca;
    private JTextField campoMarca;
    private JTextField campoColor;
    private JTextField campoTipo;
    private JTextField campoEspacio;
    private JButton botonGuardar;

    private Cliente propietario;

```

```java
private void guardarVehiculo() {
    try {
        String placa = campoPlaca.getText().trim();
        String marca = campoMarca.getText().trim();
        String color = campoColor.getText().trim();
        String tipo = campoTipo.getText().trim();
        String espacio = campoEspacio.getText().trim();

        if (placa.isEmpty() || marca.isEmpty() || color.isEmpty() || tipo.isEmpty() || espacio.isEmpty()) {
            JOptionPane.showMessageDialog(this, "Debe completar todos los campos.");
            return;
        }

        Vehiculo vehiculo = new Vehiculo(placa, marca, color, propietario, tipo);
        vehiculo.asignarEspacio(espacio);
        vehiculo.registrarIngreso(LocalTime.now());

        propietario.setVehiculo(vehiculo);

        Connection conn = Conexion.conectar();
        String sql = "INSERT INTO vehiculo (placa, marca, color, propietario, tipo, espacioAsignado, horaIngreso) " +
                     "VALUES (?, ?, ?, ?, ?, ?, ?)";
        PreparedStatement stmt = conn.prepareStatement(sql);
        stmt.setString(1, vehiculo.getPlaca());
        stmt.setString(2, vehiculo.getMarca());
        stmt.setString(3, vehiculo.getColor());
        stmt.setString(4, vehiculo.getPropietario().getNombre());
        stmt.setString(5, vehiculo.getTipo());
        stmt.setString(6, vehiculo.getEspacioAsignado());
        stmt.setString(7, vehiculo.getHoraIngreso().toString());

        stmt.executeUpdate();
        conn.close();

        JOptionPane.showMessageDialog(this, "Vehículo guardado correctamente.");
        this.dispose();

    } catch (SQLException ex) {
        ex.printStackTrace();
        JOptionPane.showMessageDialog(this, "Error al guardar el vehículo en la base de datos.");
    }
}
```

8. Pruebas

8.1. Casos de Prueba Realizados

Se realizaron pruebas manuales a través de la interfaz gráfica para validar las funcionalidades clave.

- **Registro de Cliente:**
  - **CP-01:** Ingresar un nombre válido para el cliente. (Resultado: Cliente creado y formulario cerrado).
  - **CP-02:** Intentar guardar un cliente con el campo de nombre vacío. (Resultado: Mensaje de error "Debe ingresar un nombre.").
- **Registro de Vehículo y Entrada:**
  - **CP-03:** Ingresar datos válidos para un vehículo y asignarlo a un cliente recién creado. (Resultado: Vehículo guardado en DB y formulario cerrado).
  - **CP-04:** Intentar guardar un vehículo con campos vacíos. (Resultado: Mensaje de error "Debe completar todos los campos.").
  - **CP-05:** (No implementado en el código, pero un caso futuro) Intentar registrar un vehículo con una placa ya existente. (Resultado esperado: Mensaje de error indicando placa duplicada).

- **Cálculo de Tarifa y Salida:** (Estas funcionalidades son manejadas por ControlIngreso pero no expuestas directamente en la GUI proporcionada. Se asume que serían invocadas por una futura interfaz o lógica de negocio).
    - **CP-06:** Simular una entrada y salida con duración conocida para verificar calcularTarifa().

9. Conclusiones

Este proyecto ha logrado implementar un sistema de gestión de parqueo fundamental utilizando los principios de la programación orientada a objetos y una interfaz gráfica básica.

- **Reflexión sobre el uso de patrones y cómo facilitaron el diseño:**
    - La implementación del patrón **Listener (Callback)** demostró ser una solución efectiva para la comunicación entre componentes de la interfaz gráfica (ClienteForm y VehiculoForm) de manera desacoplada. Esto facilita la extensión de la lógica de flujo de la aplicación sin modificar directamente el código de los formularios, permitiendo que un formulario notifique eventos sin conocer los detalles de cómo esos eventos serán manejados.
    - Aunque otros patrones como Singleton, Strategy o Factory Method no se evidencian explícitamente en el código base proporcionado, el diseño de clases como Parqueo (con su potencial para ser Singleton) y Tarifa (con calcularTarifa que podría usar Strategy) demuestra la aplicabilidad de estos principios en futuras iteraciones.
- **Ventajas observadas en la estructura del sistema:**
    - **Modularidad:** Las clases están bien definidas con responsabilidades específicas (ej. Conexion para la BD, UbicacionParqueo para los espacios).
    - **Interfaz de Usuario:** La inclusión de Swing mejora la interacción del usuario significativamente en comparación con una aplicación de consola.
    - **Persistencia de Datos:** La conexión a MySQL asegura que los datos sean persistentes y gestionables en un entorno de base de datos robusto.
- **Qué se aplicó de los contenidos de la materia:**
    - **Programación Orientada a Objetos:** Uso de clases, objetos, atributos, métodos, encapsulamiento.
    - **Modelado de Clases:** Aunque la jerarquía de Vehiculo es implícita, el modelado de las entidades (Cliente, Vehiculo, Parqueo, etc.) es claro.
    - **Manejo de Colecciones:** Uso de ArrayList para gestionar listas de objetos.
    - **Eventos y Listeners:** En el diseño de la GUI para la interacción entre formularios.
    - **Persistencia:** Implementación básica de JDBC para interacción con MySQL.
    - **Manejo de Errores:** Utilización de try-catch para excepciones de SQL y validaciones básicas de entrada.
- **Qué se podría mejorar o ampliar:**
    - **Separación de Capas (DAO):** La lógica de persistencia (SQL) está actualmente acoplada a VehiculoForm. Se recomienda crear una capa DAO (Data Access Object) separada (ej. VehiculoDAO) para abstraer las operaciones de base de datos del código de la interfaz de usuario.
    - **Gestión de Excepciones:** Definir y lanzar excepciones personalizadas (ej. VehiculoYaExisteException, ParqueoLlenoException) para una gestión de errores más robusta y significativa.
    - **Jerarquía de Vehículo:** Implementar explícitamente la jerarquía de Vehiculo (ej. VehiculoEstandar, VehiculoEspecial como subclases) con polimorfismo en métodos como calcularTarifa o mostrarDatos.

- **Patrones de Diseño Adicionales:** Incorporar patrones como Singleton (para Parqueo o Conexion), Strategy (para diferentes cálculos de Tarifa), o Factory (para crear diferentes tipos de Vehiculo).
- **Funcionalidades Completas:** Ampliar la GUI para incluir la funcionalidad de salida de vehículos, consulta de vehículos estacionados y disponibilidad de espacios.
- **Seguridad:** Mejorar la seguridad de la conexión a la base de datos (ej. no dejar la clave en blanco).
- **Validaciones Más Robustas:** Implementar validaciones más complejas (ej. formato de placa, unicidad de placa antes de guardar).

10. Anexos

Código Fuente:

Ubicación: jdbc:mysql://localhost:3306/parqueo_inteligente?useSSL=false&allowPublicKeyRetrieval=true

GitHub: https://github.com/dey-h/Proyecto-121.git