TECNICATURA UNIVERSITARIA
EN PROGRAMACIÓN
A DISTANCIA

UTN
UNIVERSIDAD TECNOLÓGICA NACIONAL

Trabajo Práctico Integrador

Programación II

Sistema de Gestión de Vehículos y Seguros Vehiculares

Alumnos:

- Gabriel Valdez Arce - valdezarcegabriel@gmail.com
- Nicolas Daniel Bressan - nicolasbressan00@gmail.com
- Franco Bertotti – francob1997@gmail.com
- Daiana Judith Velasquez Torrez - velasquezdaiana08@gmail.com

Link video: link

1. Integrantes y roles del equipo

El trabajo fue desarrollado por un equipo de cuatro integrantes, y cada uno asumió distitas responsabilidades para cubrir todas las etapas del proyecto.

- **Gabriel:** diseño y creación de la base de datos, configuración de conexión JDBC y desarrollo de DAOs y consultas SQL.
- **Franco:** soporte en la construcción del modelo relacional, estructura de tablas y validaciones.
- **Nicolás:** desarrollo del menú de consola, desarrollo de la capa de servicios y lógica de interacción con el usuario.
- **Daiana Velásquez:** elaboración del diagrama UML, desarrollo de pruebas funcionales (testing) , documentación del proyecto y redación del informe final.

La colaboración fue dinámica y todos participamos en varias tareas, estos roles representan las contribuciones principales de cada miembro.

2. Elección del dominio y justificación

El dominio seleccionado fue **Vehículo – Seguro Vehicular**, una relación frecuente en sistemas administrativos reales, como aseguradoras, registros automotores o flotas empresariales.

La elección se fundamenta en:

- Permite modelar una **relación 1→1 clara** (un vehículo tiene un único seguro activo).
- Proporciona atributos variados (fechas, cadenas, enteros, enumeraciones) qulo que facilita la pràctica de validaciones y mapeo a la base de datos.
- Permite implementar un sistema CRUD completo con dos entidades relacionadas.
- Es un dominio comprensible para cualquier usuario, lo que facilita pruebas y verificaciòn del sistema.
- La relaciòn 1 --> 1 permite aplicar reglas de negocio simples pero reales, como impedir que un seguro sea asignado a más de un vehìculo.

3. Diseño general y UML

Se estableció que :

- A= Vehículo
- B= SeguroVehicular

La relación establecida es **1 → 1 unidireccional**, lo que significa que:

- **Vehiculo conoce a su SeguroVehicular** mediante el atributo seguro.
- **SeguroVehicular NO referencia al Vehiculo**, manteniendo una dependencia mínima.

Decisión clave de diseño de la relación 1 --> 1

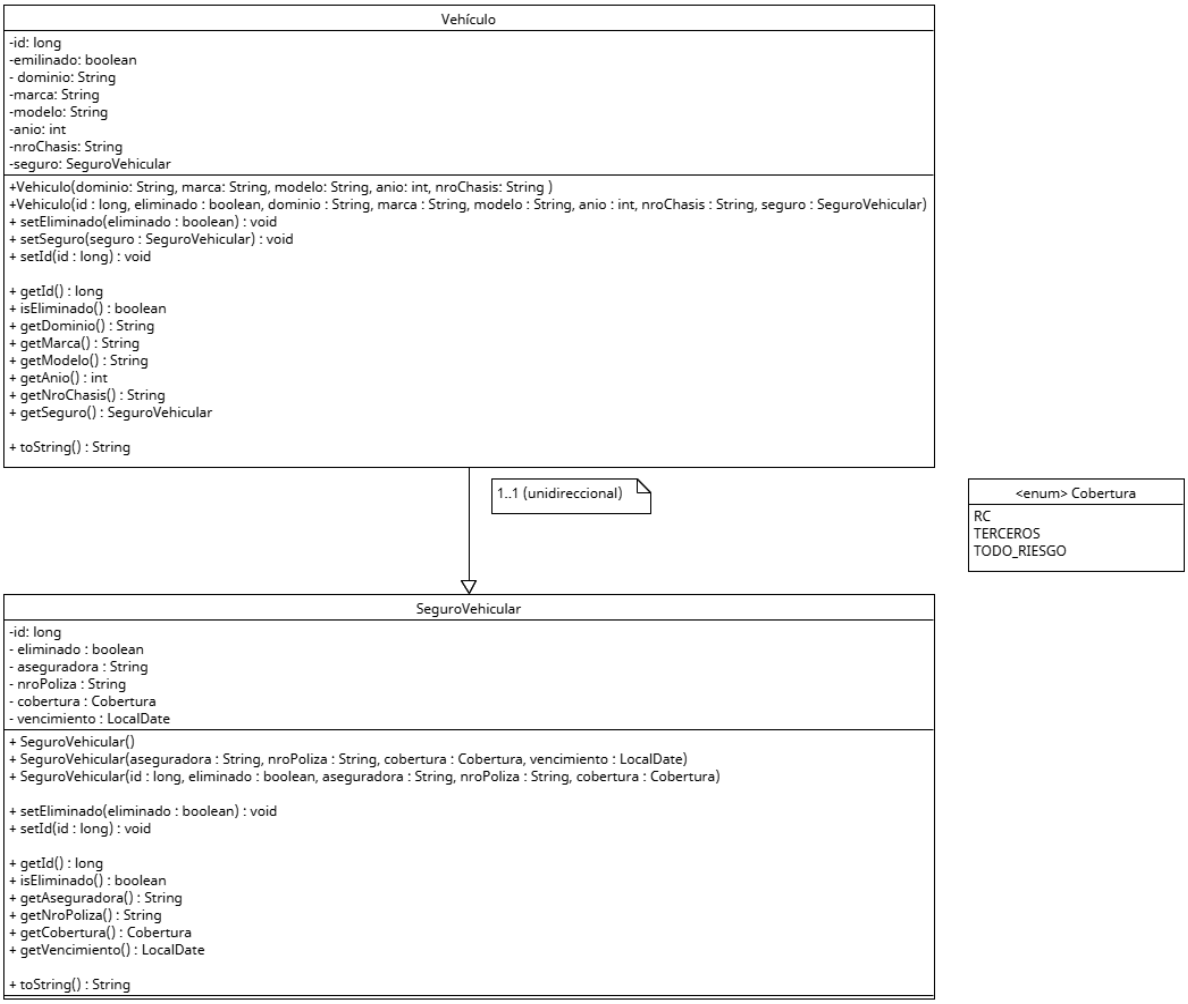
Se implementó mediante **:**

Foreign Key UNIQUE en la tabla Vehiculo --> Vehiculo.Seguro

Esto garantiza que un seguro no pueda asignarse a más de un vehículo.

→ **El diagrama UML de clases representa el modelo de dominio del sistema.**
En él se muestran las entidades principales: *Vehiculo* y *SeguroVehicular*, junto con sus atributos, constructores y métodos públicos.

→ También se incluye el enumerado *Cobertura*, utilizado para representar el tipo de protección del seguro (RC, TERCEROS, TODO_RIESGO).



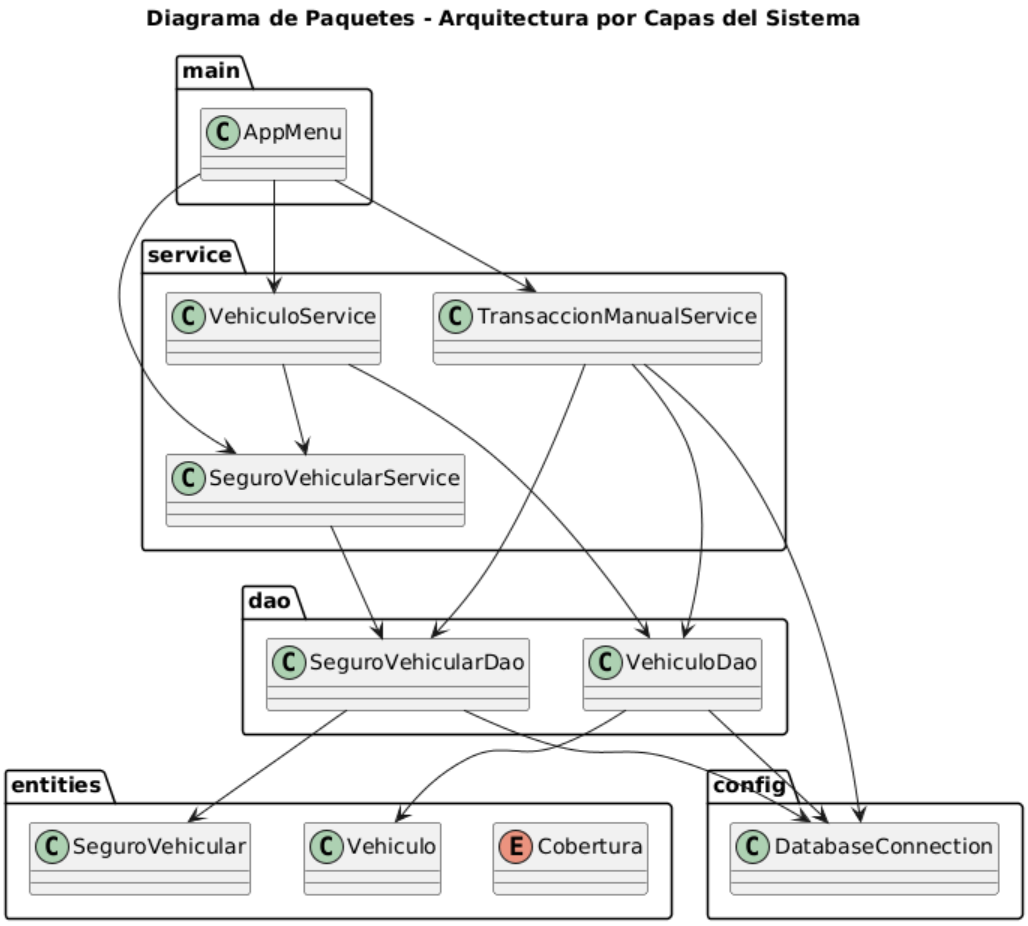
→ **El diagrama de paquetes muestra la arquitectura por capas del sistema**, organizada según la estructura solicitada por la cátedra.

→ Cada paquete agrupa clases con responsabilidades específicas:

- **config/**: manejo de la conexión a la base de datos.
- **entities/**: clases del dominio (Vehiculo, SeguroVehicular, Cobertura).
- **dao/**: acceso a datos con JDB
- **service/**: lógica de negocio, validaciones y transacciones (commit/rollback).
- **main/**: entrada de la aplicación y menú de consola.

El diagrama también muestra las **dependencias entre capas**:

- Los **DAO dependen de entities y config** para interactuar con la base.
- Los **Service dependen de los DAO** para aplicar reglas y validaciones.
- El **menú (AppMenu)** invoca a los services para ejecutar las operaciones del usuario.
- La clase **TransaccionSeguraService** encapsula la operación compuesta con transacciones manuales.



Este diagrama permite visualizar claramente cómo se organiza y comunica cada parte del sistema, asegurando una arquitectura modular y mantenible.

4. Arquitectura por capas

Config/

Contiene la clase DatabaseConnection, responsable de gestionar la conexión con la base de datos.

Centraliza la configuración JDBC y permite obtener una conexión reutilizable para todas las operaciones del sistema.

entities/

Incluye las clases de dominio:

- Vehículo
- SeguroVehicular

- Cobertura (Enum)

En estas clases representan los objetos reales del modelo y definen sus atributos, constructores y métodos asociados.

dao/

Implementa el patròn DAO (Data Acess Object) para interactuar directamente con la base de datos mediante JDBC.

- GenericDao <T>
- VehiculoDao
- SeguroVehicularDao

Todas las operaciones usan PreparedStatement.

service/

Esta capa contiene la lógica de negocio del sistema.

- VehiculoService
- SeguroVehicularService
- TransaccionManualService

Main/

Contiene la interfaz de usuario por consola (AppMenu), que invoca los servicios para ejecutar las acciones solicitadas por el operador.

Esta capa no interactúa directamente con la base de datos, sino a través de la capa service.

5. Persistencia y Base de Datos (MySQL)

A continuación se muestra el script SQL utilizado para la creación de las tablas:

5.1 script SQL :

```
CREATE TABLE vehiculo (
 id INT AUTO_INCREMENT PRIMARY KEY,
 eliminado TINYINT NOT NULL,
 dominio VARCHAR(10) NOT NULL,
```

```
 marca VARCHAR(50) NOT NULL,
 modelo VARCHAR(50) NOT NULL,
 anio INT NOT NULL,
 nroChasis VARCHAR(50) NOT NULL,
 seguro INT UNIQUE,
 FOREIGN KEY (seguro) REFERENCES segurovehicular(id)
);

CREATE TABLE segurovehicular (
 id INT AUTO_INCREMENT PRIMARY KEY,
 eliminado TINYINT NOT NULL,
 aseguradora VARCHAR(80) NOT NULL,
 nroPoliza VARCHAR(50) NOT NULL UNIQUE,
 cobertura VARCHAR(15) NOT NULL,
 vencimiento DATE NOT NULL
);
```

5.2 Relación 1 --> 1

```
ALTER TABLE vehiculo
ADD CONSTRAINT uq_seguro UNIQUE (seguro);
```

Un seguro pertenece a un único vehículo
Un vehículo tiene un único seguro asignado

5.3 Manejo de transacciones

En este proyecto implementamos dos enfoques de manejo de transacciones dentro de la capa service/.

- **Transacciones automáticas (autoCommit = true)**
  Este es el comportamiento por defecto de JDBC.
  Los servicios principales (VehiculoService y SeguroVehicularService) trabajan con:
  - Conexiones individuales y autoCommit activado, donde cada operación SQL se confirma automáticamente.
- **Transacciones manuales (autoCommit = false)**
  Para cumplir con el punto obligatorio del enunciado, se incorporó una clase específica:
  TransaccionManualService en /service.
  Esta clase implementa una transacción compuesta manual, utilizando:
  -una única conexión compartida
  -autoCommit(false) para iniciar la transacción
  - commit() si todas las operaciones finalizan correctamente
  -rollback() ante cualquier excepción o fallo.

  Operación implementada: crear vehículo + crear seguro + asociarlos (todo en un único paso atómico)

  Crea el seguro --> crea el vehículo - -> asocia el seguro al vehículo

  Y solo si las 3 operaciones salen bien --> **commit**

  si algo falla en algún punto --> **rollback.**

  La coexistencia de dos enfoques de transacciones es intensional:

  - Las operaciones simples continúan usando transacciones automáticas (más legible y suficiente)
  - Las operaciones que involucran más de una tabla (regla 1-->1) se manejan con una transacción manual para cumplir los requisitos del proyecto.

6. Validaciones y reglas de negocio aplicadas

La lógica de negocio del sistema se implementa principalmente en la capa **Service** y en el **menú de consola**, donde se realizan validaciones esenciales para garantizar que los datos ingresados sean correctos y que las operaciones respeten las reglas de dominio.

Validaciones implementadas:

- Conversión del dominio a mayúsculas.

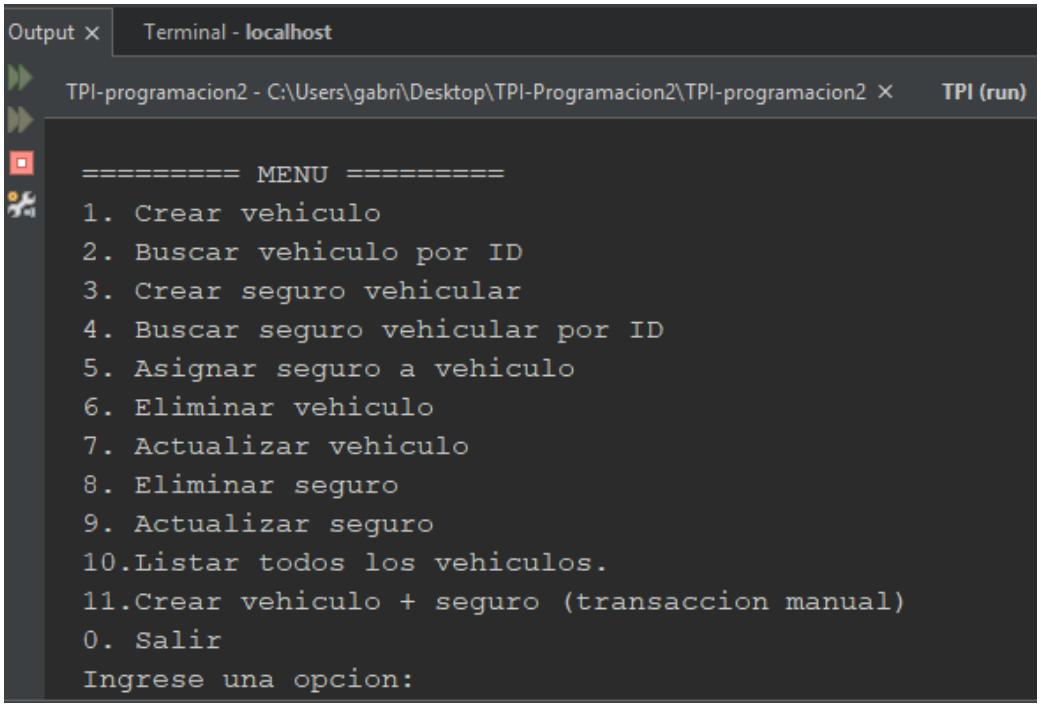
- Control de excepciones por entradas inválidas en el menú.
- Validación de formato de fecha.
- Control de IDs no numéricos.
- Validación de existencia de vehículos antes de asignarle seguro.
- Validación previa de existencia del seguro antes de asociarlo.
- uso de enumerado (Cobertura), evitando el ingreso de valores incorrectos.

Reglas de negocio aplicadas:

- Relación unidireccional 1→1 estricta.
- Baja lógica del vehículo (eliminado = 1).
- Regla de integridad en transacciones compuestas.
- Menú interactivo con operaciones CRUD completas, permitiendo gestionar ambas entidades desde la consola.

7. Pruebas realizadas (capturas del menú y consultas SQL útiles).

Menu:



Prueba 1: Creacion de Vehículo:

```
--- CREAR VEHICULO ---
Dominio (patente): ngf782
Marca: toyota
Modelo: etios
Anio: 2013
Numero de chasis: ch000001
Vehiculo creado con exito.
id= 4, eliminado= false, dominio= NGF782, marca= toyota, modelo= etios, anio= 2013, nroChasis= ch000001, {Datos del seguro: null}

========= MENU =========
```

8. Conclusiones

El desarrollo de este sistema permitió aplicar de manera práctica los conceptos fundamentales de programación orientada a objetos, acceso a bases de datos y arquitectura por capas. A través de las entidades Vehículo y SeguroVehicular se modeló correctamente una relación unidireccional 1→1, respetando las reglas del dominio mediante validaciones simples pero efectivas.

El uso de JDBC brindó una comprensión más profunda del manejo directo de la base de datos: conexiones, consultas preparadas, ejecución de operaciones CRUD y gestión segura de datos. El patrón DAO permitió separar claramente la responsabilidad de acceso a datos, mientras que la capa Service organizó la lógica de negocio, facilitando la lectura, mantenimiento y escalabilidad del sistema.

La incorporación de una **transacción manual (commit/rollback)** permitió simular una operación compuesta realista y asegurar la atomicidad entre la creación del vehículo y la creación del seguro asociado, mostrando cómo se controlan operaciones críticas cuando intervienen varias tablas.

El menú de consola sirvió como interfaz sencilla para interactuar con el sistema, validar las operaciones y realizar pruebas funcionales completas. En conjunto, el proyecto consolidó conocimientos de UML, diseño modular, validaciones, manejo de errores y transacciones, dando como resultado una aplicación funcional y bien estructurada.

Mejoras futuras

A futuro, el sistema podría seguir creciendo. Algunas mejoras posibles serían agregar validaciones más completas (como verificar mejor los datos del vehículo o evitar duplicados), implementar una interfaz más amigable para el usuario, o incluso transformar la aplicación en una versión web.
También sería útil mejorar el manejo de errores, generar reportes más detallados (por ejemplo, seguros próximos a vencer) y sumar pruebas automatizadas para asegurar que todo funcione correctamente.

Estas mejoras permitirían que el sistema sea más robusto, cómodo de usar y fácil de mantener.

9. Bibliografía y herramientas utilizadas

- Documentación oficial de Java SE 21.
- Documentación JDBC (Oracle).
- MySQL 8 – Manual oficial.
- UMLetino – Generación de diagramas.
- NetBeans / IntelliJ.
- ChatGPT (asistencia técnica y guía conceptual).
- StackOverflow.
- Apuntes de la cátedra.