Trabajo Practico Integrador
Programación II

Alumnos:

- Anchorena, Tomas – Comisión N°7
- Angelelli, Rodrigo Martin – Comisión N°7
- Romero, Nazareno – Comisión N°5
- Schneider, Astrid – Comisión N°7

Link al Video: https://youtu.be/dLww6z2NEIg

Link al Repositorio: https://github.com/Astrid-Schneider/TrabajoPracticoIntegradorProgramacion

Elección del dominio y justificación

El dominio elegido es la gestión de **Mascotas y Microchips**, un caso de uso real utilizado por veterinarias, municipios y refugios para identificar animales mediante dispositivos implantables. El modelo resulta adecuado para un trabajo integrador porque permite trabajar con una relación **1 a 1**, reglas de negocio claras y validaciones coherentes (microchips únicos, mascotas sin duplicados, fechas válidas, etc.). Además, el dominio es lo suficientemente simple para implementarse en una aplicación por consola, pero lo bastante rico para aplicar conceptos centrales del curso: diseño orientado a objetos, arquitectura por capas, normalización, transacciones y persistencia con JDBC.

Diseño del sistema y decisiones clave

El diseño del sistema se estructuró alrededor de dos entidades principales provenientes del dominio: Mascota y Microchip. Ambas representan conceptos centrales y reales dentro de un sistema de registro animal basado en identificación electrónica.

Relación 1→1: análisis y decisión

Conceptualmente, cada mascota puede contar con un único microchip implantado, mientras que cada microchip solo puede asociarse a una mascota. Esta restricción natural del dominio se trasladó a la base de datos mediante una relación 1→1 opcional, ya que una mascota puede existir en el sistema antes de recibir su microchip.

Durante el diseño se evaluaron dos estrategias:

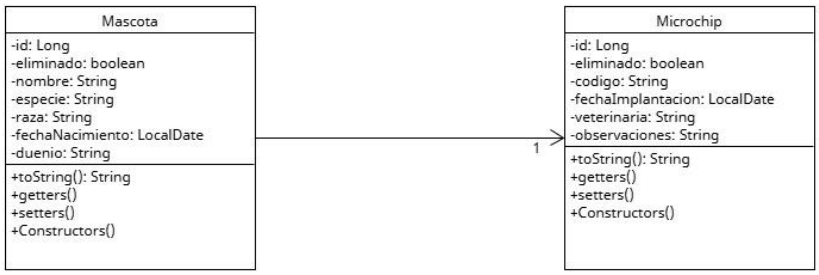
- **Clave primaria compartida**: Aprovecha la misma PK para ambas tablas. Garantiza la relación 1→1 estricta, pero acopla los ciclos de vida de las entidades y complica manejar mascotas sin microchip o microchips cargados previamente.
- **Clave foránea única (FK + UNIQUE)**: Mantiene identidades separadas para mascota y microchip. Permite mascotas sin microchip y asegura unicidad mediante una restricción UNIQUE.

Se seleccionó la estrategia FK única, aplicando sobre mascota.microchip_id la combinación FOREIGN KEY + UNIQUE. Esto logra consistencia, flexibilidad y evita duplicados.

Modelo orientado a objetos: En Java, el paquete entities contiene las clases Mascota y Microchip, ambas con atributos simples, constructores y getters/setters. La clase Mascota incluye un atributo de tipo Microchip, representando la asociación 1→1.

Se utilizaron tipos modernos (LocalDate) para fechas, mejorando claridad y evitando errores comunes con Date.

UML:



Arquitectura por capas: El proyecto sigue una arquitectura por capas, con responsabilidades claramente separadas. Esto facilita mantenimiento, legibilidad y permite futuras ampliaciones sin afectar el resto del sistema.

Capa de dominio (entities): Contiene los modelos Mascota y Microchip.
Su función es representar los datos del dominio y las relaciones entre objetos. No incluye SQL ni reglas de negocio; únicamente estado y métodos de acceso.
Esta capa es independiente del resto y puede reutilizarse en otros tipos de interfaz (por ejemplo, una API o aplicación gráfica).

Capa de acceso a datos (dao): Es responsable de toda interacción con la base de datos vía JDBC y está compuesta por:

- GenericDao<T>: estructura común para CRUD.
- MascotaDao: persistencia específica de Mascotas.
- MicrochipDao: persistencia específica de Microchips.

Principales responsabilidades:

- Ejecutar sentencias SQL (insert, update, delete, select).
- Mapear resultados hacia objetos del dominio.
- Utilizar PreparedStatement para evitar inyección SQL.
- Administrar excepciones SQL y transferirlas hacia servicios.

La capa DAO no contiene reglas de negocio y no toma decisiones de validación.

Capa de negocio (service): Implementa las reglas de negocio del sistema y coordina el trabajo de uno o varios DAOs. Aquí se define la lógica que asegura que los datos sean coherentes con el dominio e incluye:

- MascotaService
- MicrochipService
- GenericService<T>

Responsabilidades clave:

- Validar datos antes de persistir cambios.
- Impedir duplicación de microchips o asociaciones inconsistentes.
- Gestionar operaciones que involucran múltiples entidades.
- Controlar transacciones: decidir cuándo aplicar commit o rollback.

La presentación nunca interactúa directamente con la base de datos; siempre lo hace mediante esta capa.

Capa de infraestructura (config): El paquete config incluye la clase DatabaseConnection, encargada de:

- Establecer la conexión JDBC.
- Mantener credenciales y URL de conexión.
- Permitir la activación/desactivación de autoCommit.
- Facilitar la obtención de conexiones para DAOs y servicios.

Esta capa centraliza la infraestructura, lo que evita duplicar código y simplifica futuros cambios (como migrar a un pool de conexiones).

Capa de presentación (main): Implementa la interfaz con el usuario mediante consola. Contiene Main y AppMenu, responsables de:

- Mostrar opciones.
- Recibir entradas del usuario.
- Validar datos básicos.
- Delegar operaciones a los servicios.
- Mostrar resultados y mensajes de error.

La capa de presentación no debe contener SQL ni decisiones de negocio.

Flujo general entre capas:

1. El usuario selecciona una operación en el menú.
2. El menú llama al servicio correspondiente.
3. El servicio aplica reglas de negocio y llama al DAO.
4. El DAO ejecuta SQL en la base de datos y retorna resultados.
5. El servicio procesa la respuesta y la devuelve al menú.
6. El usuario visualiza el resultado.

Esta arquitectura permite un sistema modular, escalable y mantenible.

Persistencia, estructura de la base y manejo de transacciones: La persistencia del sistema se implementó utilizando JDBC puro, con consultas SQL definidas en las clases DAO y gestionadas a través de la clase DatabaseConnection. El diseño busca mantener la consistencia del dominio Mascota–Microchip y garantizar que las operaciones compuestas se ejecuten de manera atómica.

Estructura de la base de datos: El sistema utiliza dos tablas principales:

- microchip (id, codigo, fecha_implantacion, veterinaria, observaciones, eliminado)
- mascota (id, nombre, especie, raza, fecha_nacimiento, duenio, microchip_id, eliminado)

Las claves primarias son autoincrementales; ambas tablas incluyen el campo eliminado para implementar baja lógica. Esto permite mantener históricos y evita errores por restricciones FK.

Relación 1→1 implementada: La tabla mascota contiene la FK microchip_id, declarada como:

- FOREIGN KEY hacia microchip(id)
- UNIQUE, lo que asegura que un microchip no pueda asociarse a más de una mascota.

Esta estructura refleja fielmente la relación 1→1 opcional del dominio.

Restricciones relevantes

- UNIQUE(codigo) en microchip.
- NOT NULL en atributos obligatorios (nombre, especie, duenio, codigo).
- CHECK para validar longitudes y garantizar consistencia (por ejemplo, no admitir fecha de implantación sin veterinaria).

Estas validaciones en SQL complementan las realizadas desde Java y ayudan a mantener integridad aún frente a operaciones incorrectas.

Orden de operaciones en la persistencia: El flujo típico de una operación depende del tipo de entidad involucrada:

- **Alta de mascota sin microchip:**
  - El usuario ingresa datos.
  - MascotaService valida campos obligatorios y fechas.
  - MascotaDao.save() inserta la mascota sin microchip_id.
- **Alta de microchip:**
  - El usuario ingresa el código y datos opcionales (veterinaria, fecha).
  - MicrochipService verifica que el código no exista previamente.
  - MicrochipDao.save() inserta el nuevo microchip.
- **Asociar un microchip a una mascota:**
  - El servicio verifica que:
    - La mascota no tenga ya un microchip.
    - El microchip no esté asociado a otra mascota.
  - Se actualiza la tabla mascota con el microchip_id.
  - El servicio confirma la operación o revierte en caso de error.
- Modificación y baja lógica:
  - Las actualizaciones se realizan mediante métodos update() en cada DAO.
  - La baja lógica se implementa seteando eliminado = TRUE, sin borrar registros.

Manejo de transacciones: En JDBC, las transacciones se manejan mediante Connection.setAutoCommit(false) y los métodos commit() y rollback().
La lógica de negocio ubicada en los servicios decide cuándo iniciar o revertir transacciones.

Cuándo se utiliza transacción: Se aplica manejo transaccional en operaciones que involucran más de una acción en la base de datos, como:

- Asociar un microchip a una mascota.
- Actualizar datos de la mascota y su microchip en una misma interacción.
- Cambiar el estado lógico de ambos registros simultáneamente.

En estos casos:

- El servicio obtiene una conexión con autoCommit(false).
- Invoca los métodos DAO necesarios.
- Si todas las operaciones se ejecutan correctamente → commit().
- Si ocurre un error o una validación falla → rollback().

Validaciones y reglas de negocio El sistema incluye un conjunto de validaciones diseñadas para garantizar la coherencia del dominio Mascota–Microchip y evitar inconsistencias tanto a nivel lógico como en la base de datos. Estas validaciones se aplican principalmente en la capa service, mientras que las restricciones más estrictas se refuerzan también mediante reglas SQL (NOT NULL, UNIQUE, CHECK).

Validaciones de Mascota: Las validaciones se aplican en la creación, modificación y asociación con microchips:

- **Campos obligatorios:** nombre, especie y duenio no pueden quedar vacíos.
  Esto se controla antes de persistir la mascota mediante validaciones simples en el servicio.
- **Fecha de nacimiento válida**: Se verifica que la fecha ingresada no sea futura y tenga un formato correcto.
- **Evitar duplicación de microchips**: Si la mascota ya tiene un microchip_id, no se permite asignar otro sin antes remover la asociación.
- **Baja lógica:** Una mascota no puede eliminarse físicamente; se marca como eliminado = TRUE.
  No se permite eliminar una mascota si se detecta que el microchip continúa activo o asociado.

Validaciones de Microchip: Las validaciones buscan asegurar la unicidad del dispositivo y la coherencia de sus datos:

- **Código único**: Antes de insertar un microchip, se verifica que el código no exista previamente. Esto se controla tanto en service como mediante un UNIQUE en SQL.
- **Fecha de implantación y veterinaria**: Si se ingresa una fecha de implantación, debe existir una veterinaria asociada. La BD refuerza esta regla mediante un CHECK.
- **Longitudes y formato de cadena**: Campos como veterinaria, observaciones y codigo respetan limites definidos en la DB.Si el usuario excede esos valores, el servicio rechaza la carga.
- **Baja lógica del microchip:** Los microchips tampoco se eliminan físicamente. La baja lógica evita pérdida de datos históricos.

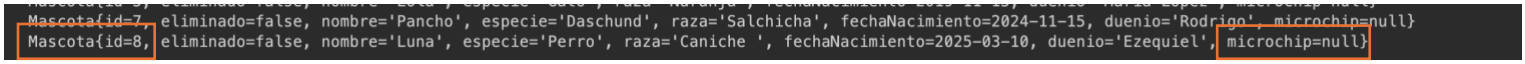
Validaciones en la asociación mascota–microchip: Esta es la parte central del sistema, ya que la relación es 1 a 1:

- **Una mascota no puede tener más de un microchip**: El servicio comprueba si la mascota ya tiene asignado un microchip_id.
- **Un microchip no puede asignarse a más de una mascota**: Antes de asociarlo, se consulta si el microchip ya se encuentra vinculado a otra mascota.
- **Consistencia transaccional**: En caso de que la asociación falle por alguna validación, la operación se revierte con rollback.

Pruebas realizadas Para verificar el correcto funcionamiento del sistema se realizaron pruebas manuales utilizando el menú por consola y consultas SQL.

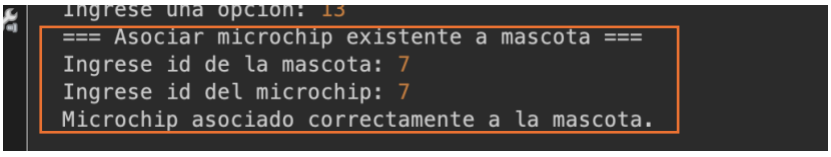
Pruebas del menú:

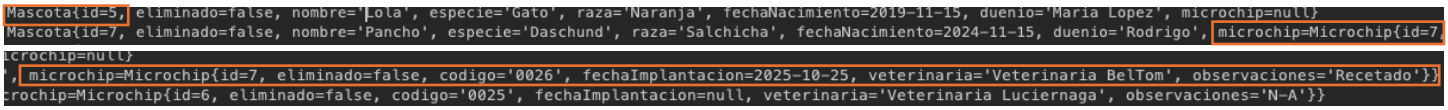
- **Alta de Mascota sin microchip**
  - Se ingresó nombre, especie, raza, fecha de nacimiento y dueño.
  - El sistema validó campos obligatorios.
  - La mascota quedó registrada con microchip_id = NULL.



- **Alta de Microchip**
  - Se ingresó el código del microchip y datos opcionales.
  - El sistema verificó unicidad del código.
  - El microchip se registró con eliminado = FALSE.

- **Asociación Mascota – Microchip**
  - Se seleccionó una mascota sin microchip.
  - Se eligió un microchip disponible.
  - El sistema verificó ambas condiciones:
    - Mascota no tiene microchip asignado.
    - Microchip no está asociado a otra mascota.





Consultas útiles en SQL

- **Consulta 1**: Listar mascotas con su microchip (si lo tienen) - Permite verificar rápidamente:
  - qué mascotas están activas, cuáles tienen microchip asignado, y ver el código y la veterinaria asociada.

```sql
USE tpibasedatos;
SELECT
    m.id,
    m.nombre,
    m.especie,
    m.duenio,
    mc.codigo AS microchip_codigo,
    mc.veterinaria AS microchip_veterinaria
FROM mascota m
LEFT JOIN microchip mc ON m.microchip_id = mc.id
WHERE m.eliminado = FALSE;
```

id	nombre	especie	duenio	microchip_codi...	microchip_veterinaria
1	Luna	Perro	Astrid	CHIP-100	Vet Centro
2	Rocky	Perro	Astrid	CHIP-300	Clinica Norte
3	Mia	Gato	Astrid	NULL	NULL
4	Toby	Perro	Juan Perez	CHIP-200	NULL
5	Lola	Gato	Maria Lopez	NULL	NULL
7	Pancho	Daschund	Rodrigo	0026	Veterinaria BelTom
8	Luna	Perro	Ezequiel	0025	Veterinaria Luciernaga
9	Lola	Perro	Felipe	0030	Veterinaria Colonial

- **Consulta 2:** Microchips y si están o no asignados a una mascota – Permite:
  - Mostrar si un microchip está libre o asignado.
  - Ayuda a comprobar que no haya microchips duplicados o mal asociados.

```sql
USE tpibasedatos;
SELECT
    mc.id,
    mc.codigo,
    mc.fecha_implantacion,
    mc.veterinaria,
    mc.observaciones,
    m.nombre AS mascota_asignada
FROM microchip mc
LEFT JOIN mascota m ON m.microchip_id = mc.id
WHERE mc.eliminado = FALSE;
```

id	codigo	fecha_implantaci...	veterinaria	observaciones	mascota_asignada
1	CHIP-100	2024-01-10	Vet Centro	Control general	Luna
2	CHIP-200	NULL	NULL	Rescatado sin datos de veterinaria	Toby
3	CHIP-300	2023-12-05	Clinica Norte	Vacunas al dia	Rocky
4	CHIP-400	NULL	NULL	Chip encontrado sin informacion	Max
6	0025	NULL	Veterinaria Luciernaga	N-A	Luna
7	0026	2025-10-25	Veterinaria BelTom	Recetado	Pancho
8	0030	NULL	Veterinaria Colonial	Implantado	Lola
9	CHIP-0019	NULL	BelTom	Sin implantar	NULL
10	CHIP-1416	NULL	Veterinaria Luro	N/A	NULL

Posibles mejoras a futuro del sistema:

Una posible mejora sería incorporar una sección de reportes y estadísticas, por ejemplo microchips activos, mascotas sin identificar o distribución por especie. También podría agregarse la funcionalidad de desasociar un microchip en caso de reemplazo o error administrativo. A nivel de arquitectura, sería útil implementar un pool de conexiones o migrar a un ORM como Hibernate para simplificar la persistencia. Otra mejora sería permitir adjuntar documentación adicional, como certificados veterinarios o historial de implantación. Agregar paginación en los listados sería conveniente para bases de datos grandes. Finalmente, una interfaz gráfica (GUI) o una API REST permitiría integrar el sistema con aplicaciones móviles o con sistemas municipales de registro animal.

Conclusión Final:

El desarrollo de este Trabajo Práctico Integrador fue una experiencia formativa muy completa para nuestro grupo, ya que nos permitió integrar conocimientos de programación en Java, modelado de bases de datos y diseño por capas en una aplicación funcional. Trabajar entre cuatro integrantes implicó coordinar agendas, dividir tareas de forma equilibrada y aprender a integrar distintas partes del sistema en un proyecto unificado. Esta dinámica grupal fortaleció nuestras habilidades de comunicación, resolución de problemas y revisión colaborativa del código.
A lo largo del proyecto, enfrentamos diversos desafíos: interpretar correctamente la relación uno a uno entre Mascota y Microchip, implementar CRUD completos usando JDBC, diseñar la estructura de la base de datos con sus restricciones y validar datos desde la capa de servicios. También aprendimos a manejar transacciones de manera segura mediante commit y rollback, asegurando que la base de datos mantenga siempre su consistencia incluso ante fallos. Resolver estos aspectos requirió investigación, pruebas, discusión técnica y ajustes en el diseño inicial.
El proyecto no solo nos ayudó a consolidar conceptos teóricos, sino que nos permitió vivenciar el ciclo completo de desarrollo de una aplicación: modelar el dominio, diseñar el UML, implementar la lógica, persistir datos, crear un menú de interacción y probar los distintos escenarios. Además, logramos identificar oportunidades de mejora que podrían incorporarse en futuras versiones, como reportes, paginación, una interfaz gráfica o integración con APIs externas.

Fuentes consultadas:

- Documentación oficial de Java SE 17 – https://docs.oracle.com/javase/
- Documentación oficial de MySQL 8.x – https://dev.mysql.com/doc/
- Oracle Java Tutorials (JDBC) – https://docs.oracle.com/javase/tutorial/jdbc/
- W3Schools SQL Reference – https://www.w3schools.com/sql/
- StackOverflow – Consulta de errores y buenas prácticas de JDBC – https://stackoverflow.com/

Herramientas utilizadas

- Java JDK 17 y Apache NetBeans IDE 21 - Lenguaje de programación y desarrollo del código.
- MySQL Server 8.0 y MySQL Workbench - Motor de base de datos relacional y admistracion.
- JDBC (MySQL Connector/J 8.x) Conexión entre Java y MySQL
- Git & GitHub Control de versiones y trabajo colaborativo

Declaración de asistencia mediante Inteligencia Artificial (IA):
Parte de la guía y asistencia teórica/técnica fue proporcionada mediante el uso de ChatGPT de OpenAI, utilizado únicamente como herramienta de apoyo educativo, manteniendo la comprensión, análisis, diseño, escritura y codificación final realizada por los integrantes del equipo.