Trabajo Final Integrador (TFI)

Alumnos

Diana Falla (diana.falla.cba@gmail.com)

Claudio Fiorito (Claudio80.cf@gmail.com)

Tecnicatura Universitaria en Programación - Universidad Tecnológica Nacional.

Programación 2

Docente Titular

Alberto Cortez

Docente Tutor

Neyén Bianchi

Contenido

1. OBJETIVO GENERAL

El presente trabajo tiene como objetivo desarrollar una **aplicación Java de consola** que modele dos clases relacionadas mediante una **asociación unidireccional 1 a 1**, implementando persistencia en **MySQL** a través de **JDBC** y el **patrón DAO**, e incorporando un control de transacciones (**commit** y **rollback**) desde la capa de servicio.

El proyecto se basa en un **sistema de gestión veterinaria**, donde cada mascota registrada puede tener asociado un único microchip identificatorio.

1.1 Elección del dominio y justificación

El dominio elegido fue *Mascota → Microchip*, ya que representa un caso real y claro de relación **uno a uno unidireccional**, donde cada mascota posee un único microchip identificatorio. Este contexto permite aplicar de forma práctica los conceptos de persistencia, integridad referencial y transacciones entre entidades relacionadas. Además, el ejemplo resulta fácilmente comprensible para el usuario final y permite extender el sistema hacia una futura gestión veterinaria más amplia (con dueños, turnos o historiales clínicos). La elección también facilita la validación de reglas de negocio concretas, como la unicidad del código de microchip y la baja lógica coordinada entre ambos registros.

2. ARQUITECTURA DEL SISTEMA

El sistema se estructura siguiendo una arquitectura por capas, que permite una separación clara de responsabilidades, mejorando la organización y mantenibilidad del código.

Capas del sistema:

- **Config:** manejo de la conexión JDBC (ConeccionBD).
- **Entities:** clases de dominio (Dueno, Mascota, Microchip).
- **DAO:** acceso a datos e implementación del patrón CrudDao<T>.
- **Service:** control de transacciones, validaciones y lógica de negocio.
- **Main:** interfaz de usuario por consola.

Esta organización sigue el patrón **multicapa**, recomendado para proyectos que integran lógica de negocio con persistencia en bases de datos.

3. MODELO DE DATOS

La base de datos utilizada es **MySQL**, bajo el nombre gestion_mascota.
El modelo relacional se compone de tres tablas principales:

- **duenio:** contiene los datos personales del dueño (DNI, nombre, apellido, teléfono, email, dirección).

- **mascota:** representa una mascota registrada, vinculada a un dueño mediante la clave foránea duenio_id.
- **microchip:** representa el chip implantado, asociado a una única mascota mediante mascota_id.

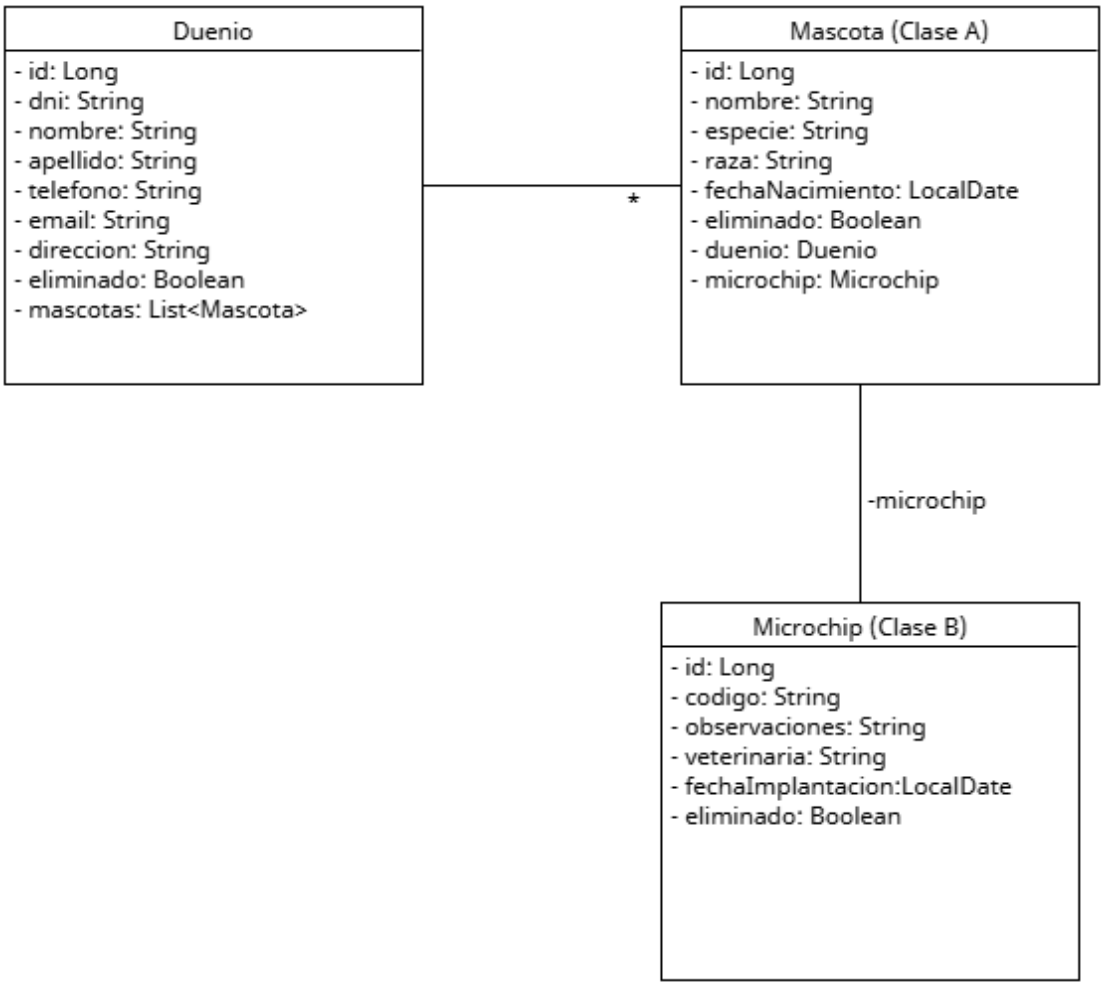
Relaciones:

- duenio (1) — mascota (N)
- mascota (1) — microchip (1)

4. DIAGRAMA UML

Dueno 1 —— N Mascota 1 —— 1 Microchip

- Un dueño puede tener varias mascotas.
- Cada mascota puede tener un único microchip asociado.
- La clase Mascota referencia a la clase Microchip (relación unidireccional 1 a 1).



5. TRANSACCIONES IMPLEMENTADAS

La clase MascotaService implementa un método denominado **crearMascotaConMicrochip()**, el cual controla la transacción completa entre las tablas mascota y microchip.

El proceso se realiza en los siguientes pasos:

1. Se inicia la conexión con setAutoCommit(false).
2. Se crea el registro de la mascota y se obtiene su ID generado.
3. Se crea el microchip asociado utilizando ese ID.
4. Si ambas operaciones se ejecutan correctamente, se realiza el commit().
5. Si ocurre cualquier error, se ejecuta el rollback() y no se persiste ningún cambio.

Esto garantiza la **integridad referencial y consistencia de datos**,

Adicionalmente, el sistema implementa bajas lógicas mediante el atributo eliminado en todas las entidades (Dueno, Mascota, Microchip).
Los registros no se eliminan físicamente de la base de datos, sino que se marca el campo eliminado como TRUE, preservando la trazabilidad de los datos.

En el script SQL, la relación 1→1 entre mascota y microchip incluye la cláusula ON DELETE CASCADE para mantener la integridad referencial; no obstante, esta eliminación física no se utiliza en la aplicación.
La baja se gestiona desde la capa de servicio, marcando ambas entidades como eliminadas cuando corresponde.

Esta decisión equilibra la integridad del modelo relacional con una gestión lógica de los datos.

6. OPERACIONES CRUD IMPLEMENTADAS

Dueños:

- Alta con validación de DNI único.
- Búsqueda por DNI o ID.
- Listado completo.
- Baja lógica (eliminado = true).

Mascotas:

- Alta (requiere dueño válido).
- Búsqueda por nombre.
- Listado completo.
- Baja lógica.

Microchips:

- Alta automática (1→1 con mascota).
- Validación de código único.
- Asociación controlada mediante transacción.

7. MENÚ DE CONSOLA

El sistema cuenta con un menú de interfaz por consola simple e intuitivo, que permite al usuario interactuar con las distintas operaciones disponibles.

```
==================================
    GESTIÓN DE DUEÑOS Y MASCOTAS
==================================
1) Crear dueño
2) Listar dueños
3) Buscar dueño por DNI
4) Crear mascota + microchip
5) Listar mascotas
6) Buscar mascotas por nombre
0) Salir
```

Cada opción del menú invoca su respectivo servicio (DuenoService o MascotaService) y maneja las excepciones mediante mensajes descriptivos.

8. VALIDACIONES PRINCIPALES

- El **DNI** del dueño es obligatorio y único.
- El **código de microchip** debe ser único.
- Una **mascota debe tener un dueño válido**.
- Se implementa **control de transacciones** (commit/rollback).
- Se utiliza **baja lógica** en lugar de eliminación física de datos.

9. ESTRUCTURA DEL PROYECTO

```
gestionmascota/
 --config/
 --entities/
 --dao/
 --service/
 --main/
```

10. SCRIPTS SQL INCLUIDOS

- **gestión_mascotas.sql**: crea la base de datos y las tablas con restricciones y claves foráneas.
- **02_datos_prueba.sql**: inserta registros iniciales de prueba (dueños y mascotas).

11. CONCLUSIÓN

El sistema desarrollado cumple con los requerimientos establecidos en el **Trabajo Integrador Final de Programación 2**, aplicando los principios de la **programación orientada a objetos**, el manejo de **persistencia con JDBC**, y el uso del **patrón DAO** para desacoplar la lógica de acceso a datos.

La implementación de transacciones garantiza la **consistencia de la base de datos**, y la estructura modular por capas facilita la mantenibilidad y escalabilidad del código.

En conjunto, el proyecto refleja la integración de los contenidos abordados en la materia, demostrando la capacidad de diseñar, implementar y documentar una aplicación completa con Java y MySQL.

12. FUENTES Y HERRAMIENTAS UTILIZADAS

Fuentes consultadas

Microsoft Learn. (2021, 12 de agosto). *Eliminación en cascada (ON DELETE CASCADE)*. https://learn.microsoft.com/es-es/ef/core/saving/cascade-delete

Stack Overflow. (2019, 13 de marzo). ¿Eliminación física o lógica de registros en base de datos? Stack Overflow. (2019, 13 de marzo). *¿Eliminación física o lógica de registros en base de datos?*

OpenAI. (2025). *Asistente ChatGPT (modelo GPT-5)*. Herramienta de apoyo utilizada para la redacción técnica, explicación de conceptos y revisión del informe final. https://chat.openai.com

W3Schools. (2024). *Tutorial de Java*. Recuperado de https://www.w3schools.com/java/default.asp

Herramientas empleadas

- **Java SE 21:** lenguaje de programación principal utilizado para el desarrollo del sistema.
- **NetBeans IDE:** entorno de desarrollo integrado para la escritura, compilación y prueba del código.
- **MySQL Workbench 8 :** diseño, modelado y gestión de la base de datos.
- **Git y GitHub:** control de versiones y almacenamiento del proyecto en un repositorio público.
- **ChatGPT (OpenAI):** herramienta de asistencia para comprender conceptos, resolver dudas y mejorar la documentación.
- **Microsoft Word y Canva:** redacción y diseño visual del informe final y diagramas.

13. LINKS

YouTube: https://www.youtube.com/watch?v=ZNZp6vacQ2E

GitHub: https://github.com/Claudio482/tp-integrador-programacion2-gestion-mascotas