TECNICATURA UNIVERSITARIA
ENPROGRAMACIÓN
A DISTANCIA

TRABAJO FINAL INTEGRADOR PROGRAMACION I

GRUPO 106

Carina Aranchet

Estefanía Avalos

Andrea Julia Ayala

Astrid Ayelén Añazco

ENLACE repositorio GitHub: Astrid97/TRABAJO-INTEGRADOR-FINAL-PROGRAMACION-II

Video: **https://drive.google.com/file/d/1SP9vuy21ff4jdE2BvHfKYklVPyD0DQLW/view?usp=sharing**

UML: **UML_TPIP2.uxf**

Introducción

El objetivo de este Trabajo Práctico Integrador es poder aplicar los conceptos aprendidos durante este curso, sobre los cuales también hemos investigado en conjunto a fin de desarrollar una aplicación en Java vinculada a la base de datos creada previamente.

Dicha base de datos corresponde a un sistema para la gestión de una cadena de clínicas veterinarias con la finalidad de analizar, administrar y registrar la implantación de chips de identificación en mascotas.

El programa en Java implementa un modelo que establece una relación unidireccional 1 a 1 entre las entidades Mascotas y Microchips, en las cuales cada mascota tiene vinculado un microchip de forma exclusiva.

Además, se incorporó una arquitectura robusta que incluye: la persistencia de datos mediate JBDC; Data Access Object (o patrón DAO) para la abstracción y encapsulación de acceso a los datos; manejo de commit y rollback para garantizar la integridad de la información en las operaciones transaccionales; y una interfaz de consola para realizar las operaciones CRUD (Crear, Leer, Actualizar, Eliminar).

Diseño de la arquitectura

De acuerdo con lo planteado en la consigna, se estructuró el proyecto con cinco capas definidas y ordenadas desde afuera hacia adentro: main, services, dao, entities, config.

Cada una de ellas tiene una función específica y a través de la interacción entre las mismas se logra la funcionalidad de la aplicación:

- Capa config: Es donde se realiza la configuración técnica y es la responsable de proveer la conexión con la base de datos de forma centralizada. Crea y devuelve un objeto Connection y, a su vez, es fundamental al momento de realizar transacciones (a través de commit si todo es correcto, o rollback ante cualquier excepción).
- Capa entities: En esta capa se representar el modelo de dominio del negocio (es decir, se muestran las cosas del mundo real con las que trabajaremos), en este caso Mascota y Microchip. Tiene como responsabilidad guardar los estados de cada entidad. No contiene lógica de acceso a datos o lógica compleja de

negocios. Además, en esta capa se incluye una clase base abstracta para todas las entidades del sistema con el fin de proporcionar campos comunes a todas las entidades, implementando el patrón de herencia para evitar la duplicación de código innecesario y soportando la eliminación lógica en lugar de la eliminación física.

- Capa dao: En esta capa, Data Access Object, como su nombre lo indica, es la encargada de acceder a la base de datos, en esta oportunidad mediante el uso de JDBC (Java Database Connectivity). La principal función es ejecutar SQL para realizar las operaciones CRUD (Crear, Leer, Actualizar, Eliminar). Está conformada por dos DAOs concretos (MascotaDaoJdbc y MicrochipDaoJdbc) y una interfaz genérica GenericDao<T>
- Capa service: Esta capa es la responsable de encapsular la lógica de negocio y coordinar los DAOs. En su estructura se ven contenidas la Interfaz GenericService<T> y las clases MascotaService y MicrochipService. Es fundamental para confirmar que se aplican las reglas de negocio, incluyendo las validaciones y combinaciones de operaciones, pudiendo invocar a varios DAOs si fuera necesario. Además, su creación permite exponer métodos claros y limpios para que puedan ser usados en la capa main.
- Capa main: Esta es la capa en la que se encuentra la interfaz que ve el usuario (en este caso, teniendo acceso por consola), por lo que es el punto de entrada. Para crearla se modelaron cuatro clases: Main, MenuDisplay y AppMenu que tiene una relación de composición con MenuHandler. Esta capa es responsable de iniciar la aplicación, permitiendo que el usuario pueda seleccionar la operación a realizar a través de un menú. Una vez seleccionada la opción, debe leerla y llamar a los métodos de la capa services.

Etapas

Para poder desarrollar este trabajo de forma ordenada, se organizó el procedimiento en cuatro etapas (cada una correspondiente a una tarea específica a realizar por cada miembro del equipo):

ETAPA 1 – Diseño y Entidades:

En esta etapa se creó el diseño UML que sirvió de estructura y guía para el resto del proyecto, determinando cuales eran las entidades, validaciones y relaciones necesarias para una correcta funcionalidad del programa completo. Además, se creó un archivo con las distintas historias de usuario que sirvieron para poder definir cuales opciones sería necesario incorporar en el menú.

ETAPA 2 – Base de Datos & Config:

En esta etapa se trabajó en las capas config y entities para lograr una correcta conexión con la Base de Datos creada en MySQL en el trabajo práctico

integrador realizado con anterioridad. Además, se modelaron las clases Microchips y Mascotas y la clase abstracta Base, que luego fueron utilizadas en la capa DAO.

ETAPA 3 – DAO (interfaces + JDBC con PreparedStatement)

Como ya sabemos, la capa DAO es la parte del sistema encargada de comunicarse directamente con la base de datos. Su responsabilidad principal es ejecutar las operaciones CRUD usando JDBC.

En este proyecto implementamos dos DAOs concretos: MascotaDaoJdbc y MicrochipDaoJdbc

Ambos siguen la interfaz genérica GenericDao<T>, que define los métodos básicos para manejar entidades.

Cada DAO se encarga de:

- Preparar las sentencias SQL necesarias
- Ejecutarlas mediante PreparedStatement
- Convertir los resultados en objetos del modelo (Mascota o Microchip)
- Manejar conexiones mediante DatabaseConnection
- Controlar errores de SQL encapsulándolos en excepciones más manejables

Además, en el caso de Mascota, el DAO también administra la relación con Microchip: si se busca, crea o elimina una mascota, se encarga también de consultar o afectar el microchip correspondiente. Esto asegura que la base de datos se mantenga consistente sin necesidad de lógica adicional en la capa superior.

ETAPA 4 – Service + Main

En esta etapa se desarrollaron dos componentes clave de la arquitectura:

<u>Capa Service – Lógica de negocio y coordinación entre DAOs</u>

La capa Service actúa como intermediaria entre el menú (main) y los DAOs, garantizando que se cumplan las reglas de negocio antes de acceder a la base de datos.
Durante esta etapa se implementaron:

- GenericService<T>: Interfaz genérica que define las operaciones CRUD de alto nivel: insertar, actualizar, eliminar, obtener por ID y listar todos.
Su existencia permite estandarizar el comportamiento de los servicios y asegurar cohesión entre las entidades.

- MascotaService: Aquí se concentra la lógica de negocio más importante del proyecto, ya que la entidad Mascota combina validaciones propias y además coordina la relación 1→1 con Microchip.

Sus responsabilidades principales fueron:

- Validar datos obligatorios de la mascota (nombre, especie, dueño).
- Manejar la creación o actualización del microchip asociado antes de persistir la mascota.
- Delegar operaciones CRUD a MascotaDaoJdbc.
- Coordinar transacciones cuando una operación requiere modificar ambas entidades.
- Garantizar la consistencia de la relación 1→1.
-

- MicrochipService: Servicio responsable exclusivamente de:

- Validar los datos del microchip.
- Encapsular la lógica antes de delegar en MicrochipDaoJdbc.
- Permitir que la capa main trabaje con un acceso limpio y controlado.

- Manejo transaccional (commit / rollback)**:** Aunque la capa Service delega las operaciones SQL a los DAOs, desde esta misma capa se coordinan las transacciones cuando ocurre una operación que afecta a ambas entidades.

Capa Main – Interfaz de usuario y orquestación de la aplicación

La capa Main es la más cercana al usuario y su responsabilidad es permitir que todas las funciones del sistema sean accesibles de manera clara desde consola.

Para ello, se implementaron cuatro clases con roles bien definidos:

- MenuDisplay: Encargada exclusivamente de mostrar el menú principal.
Para mantener la limpieza del código, no captura entradas ni realiza lógica de negocio; solo imprime las opciones.

- AppMenu: Clase que inicializa toda la aplicación y arma la cadena completa de dependencias:

DAO → Service → Handler → Menú

Sus funciones incluyen:

- Crear un único Scanner para toda la aplicación.
- Instanciar DAOs, luego Services, y finalmente MenuHandler.
- Ejecutar el ciclo principal del menú.
- Manejar entradas inválidas sin que el programa se cierre.

- MenuHandler: Es el controlador de interacción con el usuario.
Aquí se ejecuta la lógica de “flujo”:

- Capturar entradas desde consola.
- Construir objetos Mascota o Microchip.
- Aplicar reglas de actualización (Enter para mantener, creación opcional, etc.).
- Llamar a los servicios para completar las operaciones.
- Mostrar errores y resultados.

Este controlador permitió mantener el código organizado y evitar mezclar lógica de negocio con lógica de presentación.

- Main: Punto de entrada alternativo que simplemente delega inmediatamente a AppMenu, garantizando compatibilidad con cualquier IDE y ofreciendo claridad estructural.

Con esta etapa se completó la integración final entre todas las capas del sistema, logrando una aplicación totalmente operativa, robusta y alineada con las consignas del trabajo práctico.

Conclusión

En este Trabajo Práctico Integrador pudimos consolidar y aplicar de manera integral los conocimientos adquiridos en la materia, diseñando y construyendo una aplicación Java robusta que combina principios de programación orientada a objetos con técnicas avanzadas de persistencia de datos.

Partiendo del diseño inicial se fueron realizando modificaciones para optimizar el código y lograr una correcta funcionalidad de este.

Entre estas mejoras, se incluye la creación de la entidad Base que funciona como padre para manejar el ID y el estado de Microchip y Mascota a través de herencia, ya que eran atributos que en ambas entidades se repetían.

Además, se vio la necesidad de implementar el uso de una clase llamada ManageTransactions que nos permitió mantener los datos sincronizados, ya que, si los datos ingresados por consola no eran confirmados en el momento oportuno, la base quedaba desactualizada.

La arquitectura en cinco capas (main, services, dao, entities, config) fue fundamental para lograr una separación clara de responsabilidades, facilitando el mantenimiento, la escalabilidad y la testabilidad del código. La implementación del patrón DAO junto con el manejo transaccional mediante commit y rollback ha asegurado la integridad de los datos en todas las operaciones realizadas sobre la base de datos.

La relación unidireccional 1 a 1 entre las entidades Mascota y Microchip modeló eficientemente el dominio del problema, reflejando fielmente la realidad del negocio donde cada mascota posee un microchip único e identificatorio. La interfaz de consola

desarrollada proporciona una experiencia de usuario funcional para realizar todas las operaciones CRUD requeridas.

En conclusión, consideramos que en este proyecto cumple con las consignas solicitadas y representa una base sólida a la cual podrían incorporarse futuras extensiones y mejoras. La realización de este trabajo fue una gran experiencia y nos permitió reforzar y profundizar en los conceptos aprendidos, lo que constituye un valioso aporte para nuestra formación como profesionales en el campo de la programación.

CHEKLIST

UML actualizado y consistente con el código.
- ☒ A → B 1→1 garantizado por FK única o PK compartida.
- ☒CRUD de A y B con baja lógica.
- ☒ Service orquesta transacciones con commit/rollback.
- ☒DAO acepta Connection externa.
- ☒README.md con pasos para crear base y ejecutar, más enlace al video.
- ☒Archivos SQL (creación + datos) incluidos y probados.
- ☒Informe y UML dentro del repo.
- ☒Menú de consola usable y con manejo de errores.