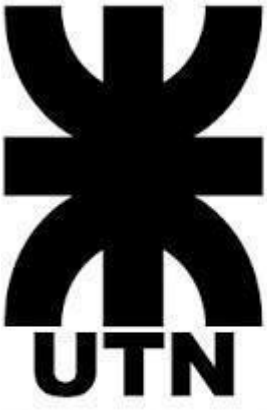
Trabajo Practico Integrador

Programación ll - Tecnicatura Universitaria en programación - UTN



1. Integrantes

-Scaglioni, Giuliano

scaglioni@gmail.com / Giuliano.scaglioni@tupad.utn.edu.ar

40089752

-Silva, Gustavo Heber

gusheber90@gmail.com

35077433

-Roure, Ricardo

Rickyroure10@gmail.com

44762754

2. Elección del Dominio y Justificación

Para este trabajo, se seleccionó el dominio **Empleado → Legajo**.

La justificación de esta elección se basa en los siguientes puntos:

Representa uno de los casos de uso más claros y realistas de una relación "uno a uno" estricta.

En cualquier organización, un empleado (la entidad principal) posee *exactamente un* legajo (la entidad dependiente) que almacena su historial y datos administrativos.

Este dominio permite implementar una variedad de validaciones como la unicidad del DNI del empleado y la unicidad del número de legajo.

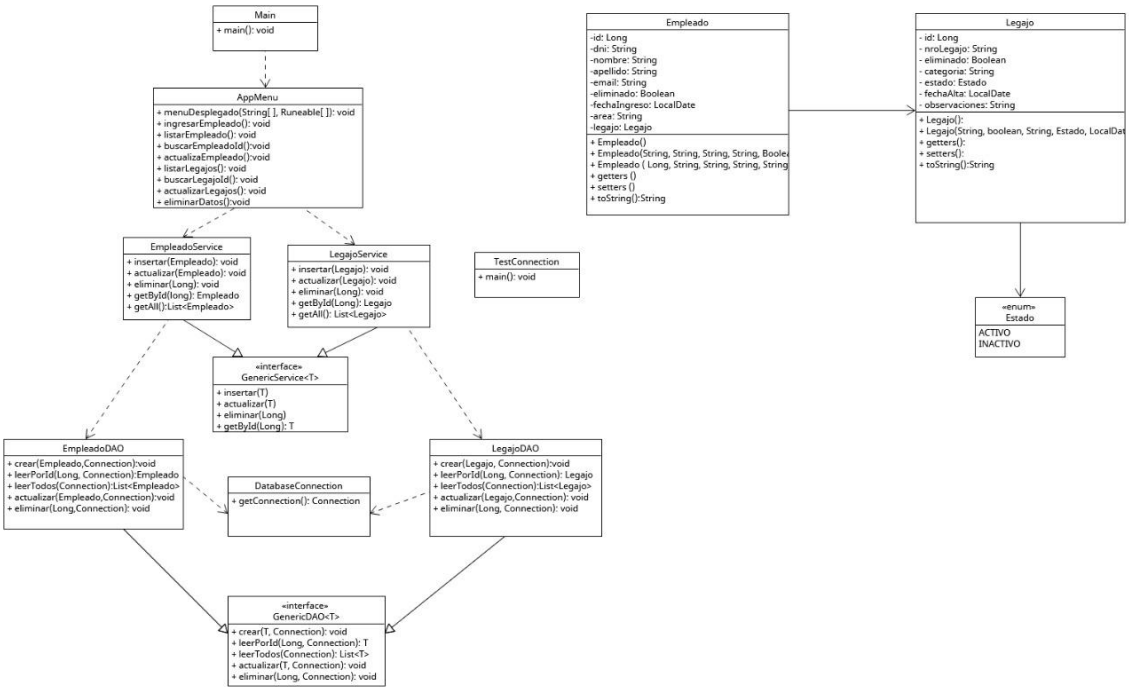
Es inaceptable que en un sistema de RRHH exista un Empleado sin su Legajo correspondiente.

Esto hace que la implementación de commit y rollback sea una necesidad crítica y no un simple ejercicio académico.

3. Diseño: Decisiones Clave y UML

Diagrama UML

A continuación, se presenta el diagrama de clases conceptual de la arquitectura diseñada.



4. Arquitectura por Capas

1. Capa de Presentación (UI de Consola)

Es la encargada de interactuar directamente con el usuario.

En este proyecto, la interfaz consiste en un menú de consola que permite seleccionar las distintas operaciones de la aplicación.

Sus responsabilidades principales son:

Recibir las entradas del usuario.

Mostrar resultados, mensajes e información relevante.

Enviar las solicitudes hacia la capa de servicios sin contener lógica de negocio.

2. Lógica de negocios (Servicios)

Actúa como intermediaria entre la UI y la capa DAO, y contiene la lógica de negocio del sistema.

En este proyecto también gestiona el control transaccional, asegurando que las operaciones que requieren acceso a la base de datos se ejecuten de manera consistente.

Responsabilidades:

Ejecutar las operaciones principales de la aplicación.

Iniciar, confirmar (commit) o deshacer (rollback) transacciones según el resultado de las operaciones.

Coordinar múltiples llamadas a DAOs cuando una funcionalidad lo amerita.

Validar datos antes de enviarlos a la capa DAO.

3. Capa de acceso a datos (DAO)

Es la capa encargada de la comunicación directa con la base de datos.

Aquí se implementa el acceso a datos mediante consultas SQL, recuperaciones, inserciones y actualizaciones.

Responsabilidades:

Gestionar las operaciones CRUD.

Ejecutar sentencias SQL utilizando conexiones provistas por la capa de servicios.

Mapear los datos obtenidos en objetos del modelo.

Aislar los detalles de persistencia para que el resto de la aplicación no dependa de ellos.

4. Capa de Base de Datos

Corresponde al motor de base de datos y su estructura interna.

Incluye las tablas, restricciones, claves foráneas y cualquier otro elemento necesario para la persistencia.

Responsabilidades:

Almacenar la información de manera persistente.

Garantizar la integridad referencial y las reglas definidas en el esquema.

Responder a las consultas realizadas desde la capa DAO

5. Persistencia

La base está compuesta por las siguientes tablas:

Tabla legajo

id: entero, clave primaria autoincremental.

nro_legajo: identificador único del legajo.

categoria: categoría asignada al legajo.

estado: estado actual del legajo (ACTIVO/INACTIVO).

fecha_alta: fecha de alta del legajo.

observaciones: comentarios adicionales.

eliminado: indicador de baja lógica.

Tabla empleado

id: entero, clave primaria autoincremental.

dni: documento del empleado, único.

nombre y apellido: datos personales del empleado.

email: correo electrónico.

fecha_ingreso: fecha de ingreso al organismo.

area: área de trabajo del empleado.

legajo_nro_legajo: número de legajo asociado, único.

eliminado: indicador de baja lógica.

Relación: legajo_nro_legajo referencia a legajo.nro_legajo, con eliminación en cascada.

Este diseño permite asegurar integridad referencial, evitando empleados sin legajo asignado y garantizando la consistencia al eliminar registros relacionados.

Orden de operaciones

Las operaciones que involucran persistencia siguen una secuencia clara que garantiza coherencia y control sobre las acciones realizadas:

La UI solicita una acción al usuario desde la consola (crear legajo, registrar empleado, modificar datos, etc.).

La capa de servicios recibe la solicitud, valida la información y prepara la operación.

Se abre una conexión a la base de datos y se inicia una transacción.

La capa de servicios invoca a los DAOs, encargados de ejecutar consultas SQL para insertar, actualizar, eliminar o recuperar datos.

Los DAOs operan directamente sobre la base, devolviendo el resultado a la capa de servicios.

Según el resultado obtenido, la operación se confirma o se revierte (ver siguiente punto).

Finalmente, la capa de servicios retorna el resultado hacia la interfaz para su visualización.

Este orden asegura que la interacción entre capas sea clara y que las operaciones críticas se realicen bajo control transaccional.

Transacciones: commit y rollback

El control transaccional se implementa en la capa de servicios, que es la encargada de coordinar las operaciones que modifican datos en la base. Allí se establece la lógica para garantizar la atomicidad de cada acción.

Se realiza un commit cuando todas las acciones del DAO se ejecutan correctamente y no se producen errores.

Se ejecuta un rollback si ocurre una excepción o si algún paso de la operación falla, asegurando que la base de datos no quede en un estado inconsistente.

De este modo, la capa de servicios actúa como punto central de control, manteniendo la integridad de las operaciones que afectan a la base y evitando resultados parciales.

6. Validaciones y Reglas de Negocio

Campos obligatorios:

- Empleado: nombre, apellido, DNI.
- Legajo: número de legajo (único), fecha de ingreso.

Regla 1 -> 1 (Empleado-Legajo):

- Se controla que cada Empleado tenga exactamente un Legajo.
- El campo empleado_id en la tabla Legajo está marcado como UNIQUE (o es la Primary Key compartida) para garantizar que un empleado no pueda tener múltiples legajos.

Baja lógica:

- Se utiliza un campo (eliminado) en lugar de borrar físicamente los registros (DELETE FROM...).
- Esto permite mantener un historial de los empleados que ya no están en la compañía y evitar la pérdida de datos.

Manejo de excepciones:

- Los DAO (EmpleadoDAO, LegajoDAO) lanzan SQLException ante errores de la base de datos (ej: violación de UNIQUE al insertar un DNI duplicado).
- La capa de Servicio (EmpleadoService) captura estas excepciones y las traduce en mensajes claros para el menú o la interfaz de usuario (ej: "Error: El DNI ingresado ya existe en el sistema").

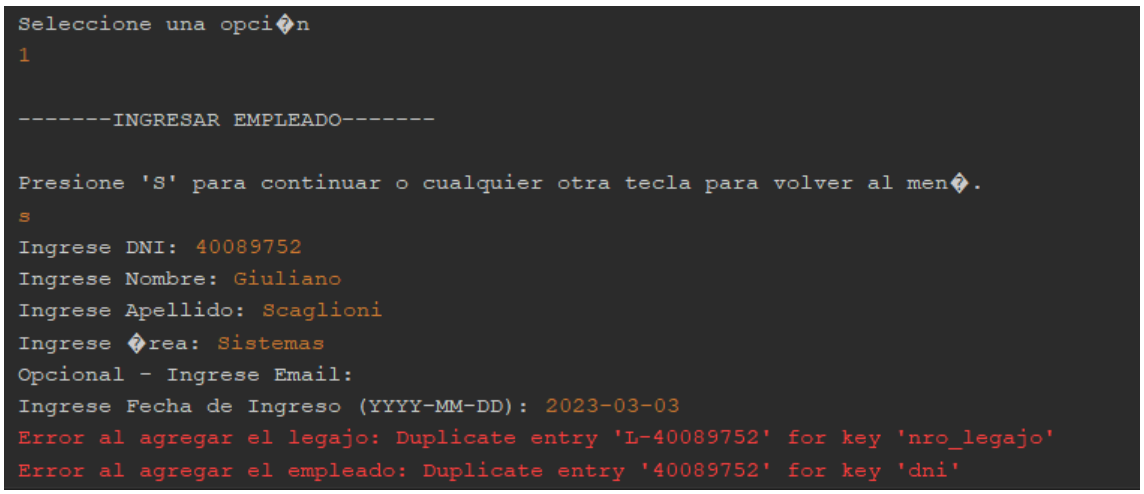
7. Pruebas Realizadas

Se implementó una interfaz de usuario basada en consola a través de un menú jerárquico y enumerado, priorizando la facilidad de uso y la eficiencia en la navegación.

El proceso de interacción se diseñó de la siguiente manera: el usuario debe seleccionar la funcionalidad deseada simplemente ingresando el número de la opción correspondiente y posterior una confirmación de que la opción sea la deseada. Tras la confirmación, el sistema procede de forma directa y solicita los datos necesarios para la operación, o bien retorna inmediatamente la información solicitada (como un listado o el resultado de una búsqueda).

Un ejemplo de prueba y error claro es;

Creamos un usuario con DNI existente.



Ingresar un empleado sin DNI:

```
Seleccione una opci□n
1

-------INGRESAR EMPLEADO-------

Presione 'S' para continuar o cualquier otra tecla para volver al men□.
s
Ingrese DNI:
Ingrese Nombre: Giuliano
Ingrese Apellido: Scaglioni
Ingrese □rea: Sistemas
Opcional - Ingrese Email:
Ingrese Fecha de Ingreso (YYYY-MM-DD): 2023
? ERROR: Formato de fecha incorrecto. Use YYYY-MM-DD.
```

Buscamos un empleado por id inexistente.

```
Empleado no encontrado
No se encontr□ un empleado con el ID especificado
Error obteniendo el legajo por ID: Cannot invoke "Models.Empleado.toString()" because "empleado" is null
? Error: Cannot invoke "Models.Empleado.toString()" because "empleado" is null
```

8. Conclusiones y Mejoras Futuras

Este proyecto demuestra una implementación sólida de los conceptos de "Programación II". Se aplicó correctamente una arquitectura por capas (separando modelo, dao, service y presentacion) y los patrones de diseño DAO y Servicios. Esta arquitectura no solo facilita el mantenimiento y la extensibilidad futura, sino que también confirma la alta comprensión de los conceptos fundamentales de la Programación Orientada a Objetos (abstracción, encapsulamiento, herencia y polimorfismo).

Técnicamente, se lograron los siguientes objetivos clave:

- Se implementó exitosamente la relación 1-a-1 unidireccional entre las entidades Empleado y Legajo, tanto en el modelo de clases Java como en la base de datos.
- La decisión de usar JDBC puro brindó un control total sobre las consultas SQL y una comprensión profunda del manejo de la persistencia.
- La gestión de transacciones (con commit y rollback) se centralizó correctamente en la capa de Servicio fue crucial para asegurar el exito de operaciones compuestas (ej: la creación de un Empleado y su Legajo asociado) y garantizar la integridad de los datos mediante *rollback* automático ante fallos.
- La persistencia en MySQL, mediante el uso de *constraints* (UNIQUE para DNI y N° Legajo, FOREIGN KEY y ON DELETE CASCADE), garantizó la integridad referencial y evitó inconsistencias.
- Se implementaron validaciones y reglas de negocio esenciales (ej: campos obligatorios, unicidad de datos). Las pruebas de consola demostraron un sistema funcional y robusto, capaz de ejecutar el CRUD completo, búsquedas y reportes.

Más allá de lo técnico, este proyecto representó una valiosa oportunidad para experimentar con decisiones de diseño reales, equilibrando simplicidad y funcionalidad. Nos ayudó a comprender la importancia de la documentación exhaustiva en un grupo de trabajo.

En síntesis, el proyecto cumple con todos los criterios de evaluación: funcionalidad, diseño por capas, calidad de código, persistencia con integridad y pruebas verificables, estando listo para su uso académico (TFI). Las limitaciones identificadas no son defectos, sino decisiones conscientes e intencionadas tomadas en este contexto, y su documentación demuestra la madurez técnica del equipo.

9. Fuentes y Herramientas Utilizadas

Para el desarrollo del Trabajo Final Integrador se utilizaron las siguientes herramientas y recursos:

Entorno de Desarrollo y Base de Datos

- Lenguaje: Java 24 JDK (recomendado por la consigna).
- IDE (Entorno de Desarrollo Integrado): Apache NetBeans (para compilación, ejecución y depuración).
- Base de Datos: MySQL 8.0.
- Cliente de BD y Diseño: MySQL Workbench (utilizado para el diseño de diagramas UML/DER, ejecución de scripts SQL de creación de tablas y pruebas de consultas).

Asistencia de IA y Redacción

- Asistencia de IA (Revisión de código, documentación y sugerencias):
    - Gemini (Google)
    - Claude (Anthropic)
    - Microsoft Copilot

Documentación y Referencias

- Material de Cátedra: Apuntes de la cátedra Programación 2 – UTN.
- Documentación Oficial:
    - Oracle. (2024). *Java Platform, Standard Edition Documentation*.
    - Oracle. (2024). *JDBC API Documentation*.
    - MySQL. (2024). *MySQL 8.0 Reference Manual*.

Link video: https://drive.google.com/drive/u/0/folders/1w_q8rCR4J9nisMIZPMYMpYS2QHVcR-PA

Link GitHub: NaD3M-71/TPI-P2