TPI Programación 2 – Grupo 7

Link video explicativo: https://drive.google.com/file/d/1F88fekAeF8Y0xy4Jv5DYID-RrxV9sEsP/view

Integrantes:

- Lautaro Ariel Cejas
- Andrés Kevin Gastaldello
- Juan Manuel Gomez
- Agustín Andrés Tejada Fernández

---

1. Introducción al Proyecto

El presente informe describe el desarrollo del **Trabajo Práctico Integrador de Programación II**, cuyo objetivo principal fue implementar una aplicación en Java que utilice una arquitectura por capas, con persistencia en base de datos relacional, relaciones entre entidades, manejo de excepciones, servicios, DAO y un menú interactivo desde consola.

El proyecto aborda un escenario aplicado donde se administra un sistema con varias entidades relacionadas. A lo largo del trabajo se buscó aplicar buenas prácticas de programación, principios de diseño y separación eficiente de responsabilidades del grupo.

El repositorio del proyecto incluye código fuente, scripts SQL, diagramas UML, pruebas en consola y toda la estructura necesaria para ejecutar y mantener el sistema hecho. La aplicación se diseñó para permitir CRUD sobre las entidades principales, validaciones, relaciones uno a uno y uno a muchos, consultas específicas y menús por consola que guían al usuario paso a paso.

---

2. Roles del Equipo

Durante el desarrollo del proyecto, cada integrante del equipo asumió funciones específicas para optimizar el flujo de trabajo y la calidad del resultado:

- **Juan realizo la etapa de entregables:** Juan asumió el rol de aglutinar todos los entregables asegurando la coherencia del proyecto, realizo el diagrama UML, el PDF y la edición del video.
- **Kevin realizo la etapa de modelado y, la etapa de implementación y carga:** Modelo las entidades que representan las tabla, creo la capa de conexión,

implemento los DAOs con CRUD, también pudo proveer un script SQL para la creación y carga de la base.

- **Lautaro realizo la etapa de seguridad y la etapa de concurrencia:** Lautaro tuvo un trabajo clave realizando los service, las transacciones, las validaciones y la integración del proyecto entero, realizando también el repositorio github.
- **Agustín manejo la etapa de consultas y reportes, desarrollando la interfaz de Consola:** El trabajo de Agustín fue esencial porque convirtió toda la lógica del proyecto en un sistema utilizable. La interfaz que creo, es la que demuestra visualmente el funcionamiento correcto de los DAOs, los servicios y la base de datos.

---

3. Arquitectura por Capas

El proyecto sigue una arquitectura clásica de **tres capas**, separando responsabilidades:

1. **Capa de Entidades (entities):** Representa las clases del modelo de dominio. Cada clase corresponde a una tabla en la base de datos.
2. **Capa DAO (dao):** Contiene las clases responsables del acceso a datos. Ejecutan consultas SQL y transforman resultados en objetos.
3. **Capa de Servicios (service):** Implementa la lógica del negocio. Llama a los DAO, valida datos, encapsula reglas y errores.
4. **Capa de Presentación:** Un menú en consola que permite al usuario interactuar con el sistema.

Esta arquitectura facilita la escalabilidad, el mantenimiento y permite reemplazar capas sin afectar a las demás.

---

4. Base de Datos y Script SQL

En la carpeta *database* se encuentra el archivo SQL utilizado para crear las tablas y relaciones.
El diseño considera claves primarias, claves foráneas, constraints y borrado en cascada donde correspondía.

Se aplicaron relaciones:

- **Uno a uno (1–1)**
- **Uno a muchos (1–N)**

Se verificó también que cada tabla estuviera normalizada y que las relaciones se correspondieran con las entidades del modelo Java.

---

5. Implementación de Entidades (entities)

En el paquete *entities* se encuentran las clases que representan las tablas.
Cada clase incluye atributos, constructores, getters, setters y métodos toString() para depuración.

Aquí se definió la relación **1 a 1**, por ejemplo:

- Una entidad A tiene una FK de B
- La relación se implementa **manteniendo la FK en una de las clases**, evitando duplicar información y asegurando integridad

---

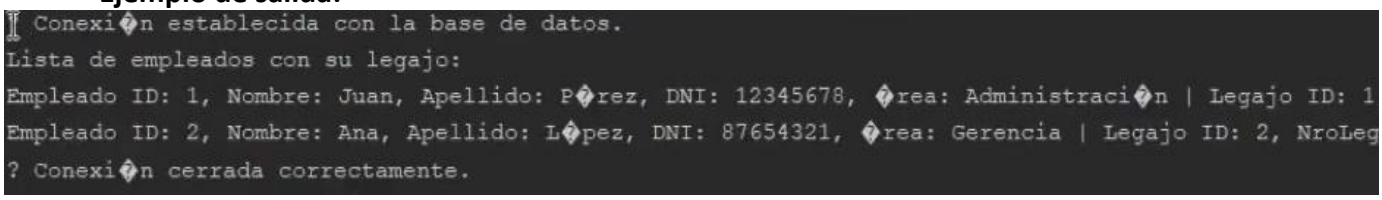
6. Capa DAO (Data Access Object)

Cada entidad tiene su correspondiente DAO, encargado de:

- Conectar a la base de datos
- Ejecutar SELECT, INSERT, UPDATE y DELETE
- Mapear los resultados a objetos Java
- Manejar excepciones SQL
- Implementar métodos como buscarPorId, listarTodos, guardar, modificar, eliminar

La separación DAO permite reutilizar código, mejorar testeo y manejar errores de forma más precisa.
También se diseñaron consultas específicas para obtener resultados más complejos cuando fue necesario.

Ejemplo de salida:



---

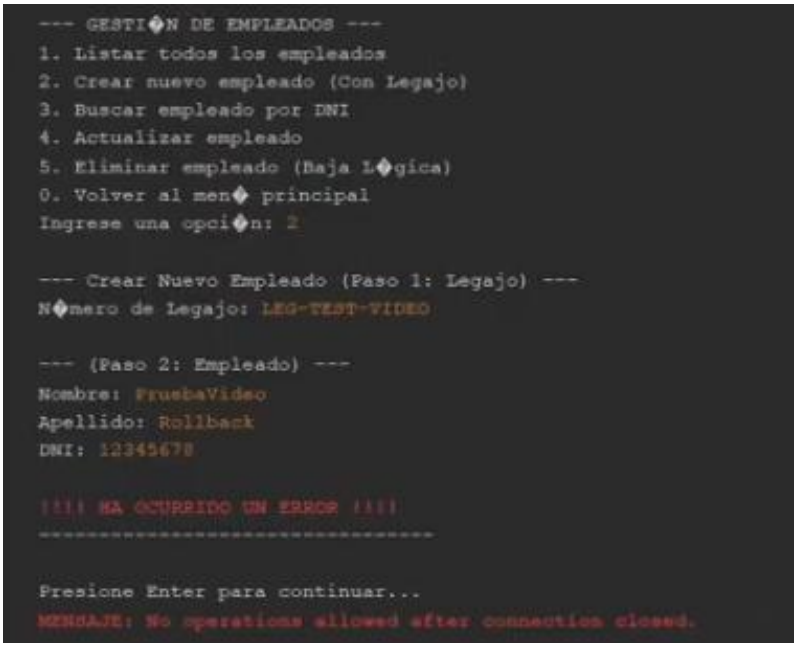
7. Capa de Servicios (service)

La capa de servicios actúa como intermediaria entre la vista y los DAO. Acá se aplican validaciones importantes, por ejemplo:

- Comprobar que no existan campos vacíos
- Verificar que los IDs ingresados correspondan a entidades reales
- Asegurar que no se rompa la relación 1–1 ni el 1–N

Los servicios representan la lógica real del programa y permiten controlar cómo se comporta el sistema frente a diferentes escenarios.

Forzamos un error para probar Rollback:



Comprobamos si se cargaron los datos aunque haya un error:

```
1 •   use empresa;
2 •   SELECT * FROM legajo WHERE nro_legajo = 'LEG-TEST-VIDEO';
3
```

Devuelve:

	id	eliminado	nro_legajo	categoria	estado	fecha_alta	observaciones
•	NULL	NULL	NULL	NULL	NULL	NULL	NULL

---

8. Menú en Consola (App/Main)

El menú principal permite navegar entre las funcionalidades del sistema.
Incluye opciones como:

- Alta de entidades
- Modificación
- Eliminación
- Consultas específicas
- Listados completos

Se utilizó un bucle while para mantener activo el menú hasta que el usuario decida salir.
La interacción incluye mensajes claros, validaciones y llamadas a los servicios correspondientes.

Ejemplo del menú eligiendo la opción 3 (buscar empleado por dni):

```
--- GESTIÓN DE EMPLEADOS ---
1. Listar todos los empleados
2. Crear nuevo empleado (Con Legajo)
3. Buscar empleado por DNI
4. Buscar empleado por ID
5. Actualizar empleado
6. Eliminar empleado (Baja Lógica)
0. Volver al menú principal
Ingrese una opción: 3
```

Ejemplo de una función realizada exitosamente:

```
--- Buscar Empleado por DNI ---
Ingrese DNI: 12345678
```

```
Empleado encontrado:
Empleado{id=1, nombre='Juan', apellido='Pérez', dni='12345678', email='juan@empresa.com', fechaIngreso=2023-01-12, area='Administración', legajo=L001,
```

Ejemplo de carga de legajo en la base de datos:

```
--- Crear Nuevo Legajo ---
Ingrese Número de Legajo: L003
Ingrese Categoria: Empleado
Ingrese Observaciones: Ninguna
Ingrese el Estado (ACTIVO / INACTIVO): activo
```

---

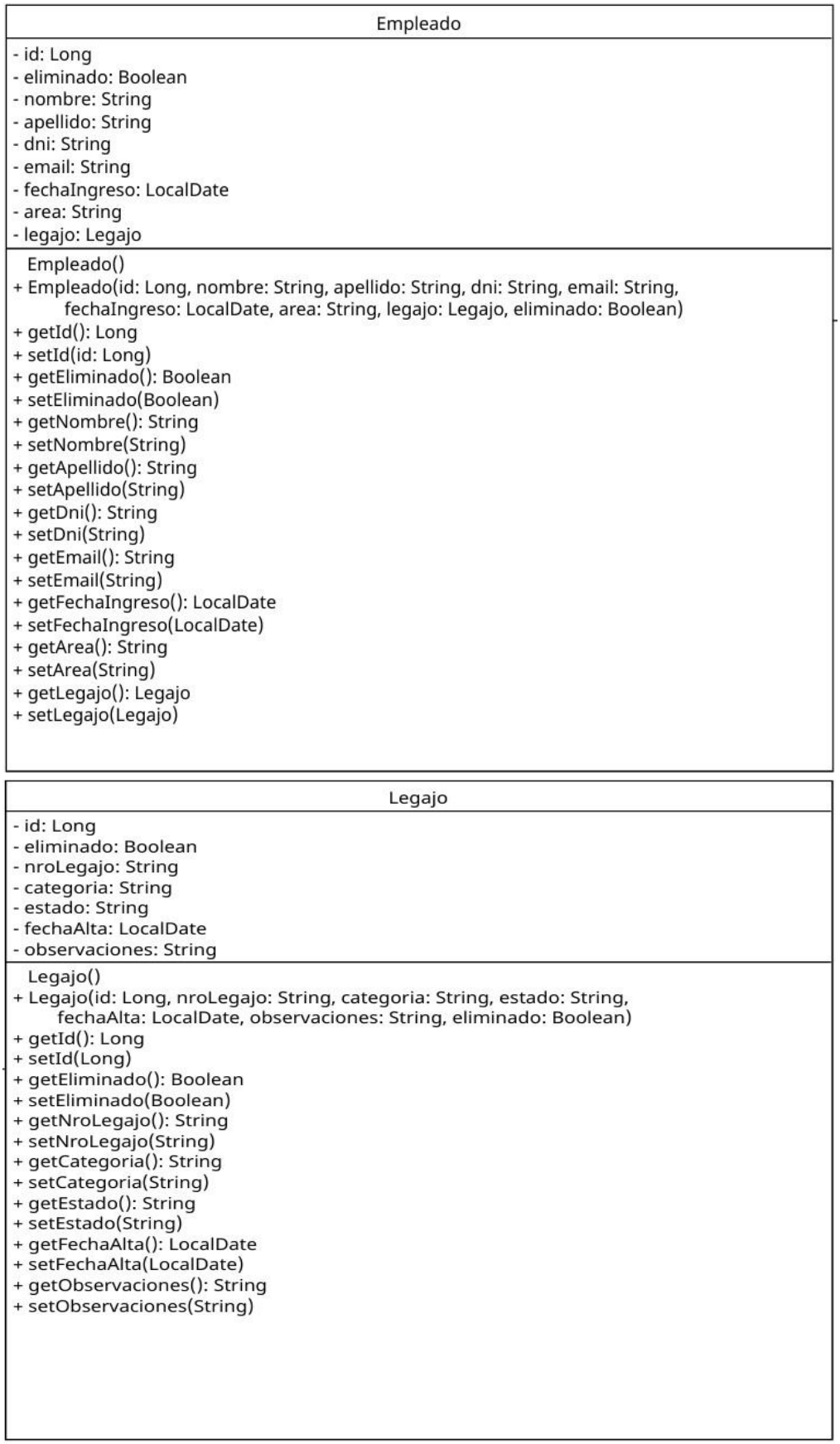
9. Diagrama UML y Coherencia con el Código

El proyecto incluye un diagrama UML ubicado en el repositorio. Este diagrama permitió definir:

- Jerarquías
- Atributos principales
- Métodos
- Relaciones entre clases

Durante el desarrollo se verificó que el código se correspondiera con el modelo conceptual.
El UML también ayudó a visualizar la relación 1–1 y su implementación concreta en Java y SQL.

UML(Empleado 1-1 Legajo):


Empleado
- id: Long
- eliminado: Boolean
- nombre: String
- apellido: String
- dni: String
- email: String
- fechaIngreso: LocalDate
- area: String
- legajo: Legajo
Empleado()
+ Empleado(id: Long, nombre: String, apellido: String, dni: String, email: String, fechaIngreso: LocalDate, area: String, legajo: Legajo, eliminado: Boolean)
+ getId(): Long
+ setId(id: Long)
+ getEliminado(): Boolean
+ setEliminado(Boolean)
+ getNombre(): String
+ setNombre(String)
+ getApellido(): String
+ setApellido(String)
+ getDni(): String
+ setDni(String)
+ getEmail(): String
+ setEmail(String)
+ getFechaIngreso(): LocalDate
+ setFechaIngreso(LocalDate)
+ getArea(): String
+ setArea(String)
+ getLegajo(): Legajo
+ setLegajo(Legajo)
Legajo
- id: Long
- eliminado: Boolean
- nroLegajo: String
- categoria: String
- estado: String
- fechaAlta: LocalDate
- observaciones: String
Legajo()
+ Legajo(id: Long, nroLegajo: String, categoria: String, estado: String, fechaAlta: LocalDate, observaciones: String, eliminado: Boolean)
+ getId(): Long
+ setId(Long)
+ getEliminado(): Boolean
+ setEliminado(Boolean)
+ getNroLegajo(): String
+ setNroLegajo(String)
+ getCategoria(): String
+ setCategoria(String)
+ getEstado(): String
+ setEstado(String)
+ getFechaAlta(): LocalDate
+ setFechaAlta(LocalDate)
+ getObservaciones(): String
+ setObservaciones(String)

11. Conclusiones del proyecto

El resultado final es un proyecto funcional, organizado, escalable y bien documentado. Además, el trabajo grupal permitió distribuir responsabilidades y combinar ideas para lograr una solución sólida. El Trabajo Práctico Integrador de Programación 2 consistió en el desarrollo completo de una aplicación Java estructurada por capas, con persistencia en MySQL mediante JDBC y el uso del patrón DAO. El proyecto se centró en modelar una relación **1 a 1 unidireccional** entre dos entidades del dominio elegido, en nuestro caso *Empleado* → *Legajo*, garantizando integridad referencial, funcionamiento transaccional, validaciones y una interfaz de consola capaz de ejecutar todas las operaciones necesarias.

A nivel conceptual, el trabajo permitió integrar varios elementos centrales del desarrollo de software profesional. En primer lugar, se modeló el dominio aplicando principios de orientación a objetos, definiendo entidades claras con responsabilidad bien delimitada y una relación estricta de dependencia donde el *Empleado* contiene la referencia a su *Legajo*. Este diseño se reflejó desde el UML hasta la estructura en código, y finalmente en el modelo físico de la base de datos mediante claves primarias y foráneas.

En la capa de datos, la aplicación utilizó JDBC de forma explícita, lo que implicó implementar consultas, inserciones, actualizaciones y eliminaciones usando PreparedStatement, asegurando seguridad, limpieza y evitando inyecciones por entrada del usuario. Los DAOs encapsularon completamente el acceso a la base, manteniendo separada la lógica de almacenamiento de la lógica de negocio.

La capa de servicios fue la encargada de coordinar las reglas del negocio, validar información crítica y operar de forma transaccional. Esto incluyó operaciones compuestas que requieren coherencia entre las dos tablas —por ejemplo, la creación de un empleado con su legajo asociado— donde el sistema debía garantizar que ambas inserciones se realicen correctamente o que ninguna se aplique en caso de error. El manejo de commit() y rollback() fue fundamental para cumplir con este requisito y para demostrar un enfoque profesional y seguro de los procesos.

Finalmente, la capa de presentación permitió que el sistema sea efectivamente utilizable. El menú de consola, desarrollado especialmente para el trabajo, organiza todas las funcionalidades del sistema en un flujo claro, intuitivo y robusto. Gracias a esta capa, se pueden crear empleados, consultar registros, listar información, actualizar datos, realizar bajas lógicas y verificar búsquedas específicas. Además, constituye la parte visible del trabajo, demostrando de forma concreta el funcionamiento de cada capa del proyecto.

En conjunto, el desarrollo logrado refleja un proceso ordenado y colaborativo, donde cada integrante del equipo asumió un rol coherente con su experiencia previa y aportó

al resultado final. Se integraron conocimientos de modelado conceptual, programación orientada a objetos, arquitectura por capas, persistencia en bases de datos, uso de transacciones y diseño de interfaces de usuario.

El proyecto permitió demostrar habilidades técnicas claves: modularización, separación de responsabilidades igualitarias, manejo correcto de excepciones, implementación de una relación unidireccional 1 a 1, claridad en el diseño UML y producción de documentación formal. Por ultimo, este TPI consolidó los contenidos vistos en las materias programación 2 y base de datos 1, logrando una aplicación completa, ordenada y funcional.