Trabajo Final Integrador (TFI)

Aplicación Java con relación 1→1 unidireccional + DAO + MySQL

Objetivo

Desarrollar una aplicación en **Java** que modele **dos clases** relacionadas mediante una **asociación unidireccional 1 a 1** (la clase "A" referencia a la clase "B"), persistiendo datos en una base relacional mediante **JDBC** y el **patrón DAO**, con **operaciones transaccionales** (commit/rollback) y menú de consola para CRUD.

Consignas generales

- Lenguaje: **Java** (recomendado 21).
- Persistencia: **JDBC** (sin ORM) con **MySQL**.
- Patrón de diseño: **DAO** y **capa Service**.
- **Código limpio**, legible, con **manejo de excepciones** en todas las capas.
- **Relación 1→1 unidireccional**: sólo la clase "A" contiene el atributo que referencia a "B".
- **Equipos de 4 personas, sin excepción**.

Dominio (elegir 1 pareja A → B)

Se debe elegir entre una de estas opciones para resolver el TPI.

- Empleado → Legajo
- Usuario → CredencialAcceso
- Vehiculo → SeguroVehicular
- DispositivoIoT → ConfiguracionRed
- Paciente → HistoriaClinica
- Empresa → DomicilioFiscal
- Libro → FichaBibliografica
- Mascota → Microchip
- Producto → CodigoBarras
- Propiedad → EscrituraNotarial
- Pedido → Envio

Estructura del proyecto (paquetes)

- `config/`: conexión a la base (lectura de propiedades externas).
- `entities/`: clases de dominio (A y B) con `id` y `eliminado` (baja lógica).
- `dao/`: interfaces genéricas y DAOs concretos (JDBC + PreparedStatement).
- `service/`: reglas de negocio, validaciones y **orquestación de transacciones**.

- `main/`: arranque de la app y **AppMenu** (consola).

Requerimientos técnicos

1) Diseño (previo al código)

- **Diagrama UML de clases** con:
  - Atributos (tipo/visibilidad) y métodos (firma/visibilidad).
  - **A → B (1..1)** unidireccional explícita.
  - Paquetes y dependencias principales.
- El UML guía toda la implementación.

2) Entidades (genérico)

- **A**: atributos propios + `id` (INT/BIGINT) + `eliminado` (BOOLEAN).
- **B**: atributos propios + `id` + `eliminado`.
- **A** contiene `private B detalle;` (o nombre equivalente).
- **Constructores** (vacío y completo), getters/setters y `toString()` legible.

3) Base de datos (MySQL)

- Clase **`DatabaseConnection`** en **`config/`** con método estático que retorne **`java.sql.Connection`**.
- Entregar un archivo **`.sql` con las instrucciones para crear la base de datos y sus tablas** (CREATE DATABASE, CREATE TABLE, claves primarias, foráneas, índices y restricciones necesarias para 1→1).
- Entregar **otro `.sql` con datos de prueba** (INSERTs) para levantar el proyecto desde cero.
- **Relación 1→1 recomendada**: **clave foránea única** en la tabla de **B** (campo `a_id` con `UNIQUE`, `FOREIGN KEY` a `A(id)`, `ON DELETE CASCADE`). Alternativa válida: **clave primaria compartida** (la PK de B es también FK a A).

4) DAO

- `GenericDao<T>` con: `crear(T)`, `leer(long id)`, `leerTodos()`, `actualizar(T)`, `eliminar(long id)`.
- DAOs concretos para A y B usando **PreparedStatement** en todas las operaciones.
- Los DAO deben ofrecer métodos que **acepten una `Connection` externa** (para participar de la misma transacción).

5) Service (transacciones obligatorias)

- `GenericService<T>`: `insertar`, `actualizar`, `eliminar`, `getById`, `getAll`.
- `AService` y `BService`:
  - **Abrir transacción**: `setAutoCommit(false)` sobre una conexión compartida.
  - Ejecutar operaciones compuestas (por ej., crear B, asociarla a A y crear A).
  - **`commit()`** si todo OK; **`rollback()`** ante cualquier error.
  - Restablecer `autoCommit(true)` y cerrar recursos.
  - **Validaciones** (campos obligatorios, formatos según dominio) y **regla 1→1** (impedir más de un B por A).

6) AppMenu (consola)

- `Main` invoca `AppMenu`.
- Convertir entradas a **mayúsculas** donde aplique.
- Debe permitir **CRUD completo** de A y B: crear, leer por ID, listar, actualizar y **eliminar lógico**.
- **Incluir al menos una búsqueda por un campo relevante (ej.: DNI, dominio, ISBN, etc., según el dominio elegido).**
- Manejo robusto de:
  - Entradas inválidas (parseos numéricos, formatos).
  - IDs inexistentes.
  - Errores de base de datos y violaciones de unicidad.
- Mensajes claros de éxito/error.

Entregables (obligatorios)

1. **Repositorio GitHub público**
   - Código fuente completo.
   - **`README.md`** con:
     - Descripción del dominio elegido.
     - Requisitos (Java/BD) y **pasos para crear la base** con el **`.sql` provisto**.
     - Cómo compilar y ejecutar (credenciales de prueba y flujo de uso).
     - Enlace al **video**.
   - **SQL**:
     - Archivo con **instrucciones para crear la base y tablas**.
     - Archivo con **datos de prueba**.
   - **UML** (imagen `.png/.jpg/.pdf`).
   - **Informe** (PDF) dentro del repo.
2. **Informe** (PDF, 6–8 páginas)
   - Integrantes (4) y roles.
   - Elección del dominio y **justificación**.
   - Diseño: decisiones clave (1→1, FK única vs PK compartida) + UML.

- Arquitectura por capas (responsabilidades de cada paquete).
- Persistencia: estructura de la base, **orden de operaciones** y **transacciones** (dónde se hace commit/rollback).
- Validaciones y reglas de negocio.
- Pruebas realizadas (capturas del menú y consultas SQL útiles).
- Conclusiones y mejoras futuras.
- **Citar fuentes y herramientas utilizadas** (incluida IA, si la usaron).

3. **Video** (10–15 minutos)
   - Presentación de los **4 integrantes** con **rostro visible**.
   - Demostración del flujo CRUD y de la **relación 1→1** funcionando.
   - Explicación **por secciones de código** (entities, dao, service, menú).
   - Mostrar una **operación transaccional** y evidenciar el **rollback** ante un fallo simulado.

Criterios de evaluación

- Correctitud funcional (CRUD, 1→1 unidireccional real en código y base).
- Diseño/arquitectura (DAO/Service, responsabilidades claras, validaciones).
- Calidad de código (legibilidad, nombres, excepciones, PreparedStatement).
- Persistencia (integridad referencial, unicidad para 1→1, scripts SQL reproducibles).
- Documentación (README claro, informe sólido, UML coherente).
- Presentación (video dentro del tiempo, explicación técnica, participación equitativa).
- Entrega (repo público completo, proyecto compilable/ejecutable desde cero).

Checklist (sugerido)

- ☐ 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.

Especificación de campos – TP Integrador (1→1 unidireccional)

Se listan los **campos obligatorios** de cada par de clases (A → B) disponible para el TP Integrador. En todos los casos:

- La relación es **unidireccional 1→1** desde **A** hacia **B** (A contiene una referencia a B).
- Ambas clases incluyen **id** (clave primaria) y **eliminado** (baja lógica).

Tip: Los nombres y longitudes son sugeridos; podés ajustarlos manteniendo la semántica, unicidad y la relación 1→1.

---

1) Empleado → Legajo

Clase Empleado (A)

Campo	Tipo (Java)	Reglas / Notas
id	Long	PK
eliminado	Boolean	Baja lógica
nombre	String	NOT NULL, máx. 80
apellido	String	NOT NULL, máx. 80
dni	String	NOT NULL, **UNIQUE**, máx. 15
email	String	máx. 120, formato email
fechaIngreso	java.time.LocalDate	
area	String	máx. 50
legajo	Legajo	Referencia 1→1 a B

Clase Legajo (B)

Campo	Tipo (Java)	Reglas / Notas
id	Long	PK
eliminado	Boolean	Baja lógica
nroLegajo	String	NOT NULL, **UNIQUE**, máx. 20
categoria	String	máx. 30
estado	Enum {ACTIVO, INACTIVO}	NOT NULL
fechaAlta	java.time.LocalDate	
observaciones	String	máx. 255

---

2) Usuario → CredencialAcceso

Clase Usuario (A)

Campo	Tipo (Java)	Reglas / Notas
id	Long	PK
eliminado	Boolean	Baja lógica
username	String	NOT NULL, **UNIQUE**, máx. 30
email	String	NOT NULL, **UNIQUE**, máx. 120
activo	Boolean	NOT NULL
fechaRegistro	java.time.LocalDateTime	
credencial	CredencialAcceso	Referencia 1→1 a B

Clase CredencialAcceso (B)

Campo	Tipo (Java)	Reglas / Notas
id	Long	PK
eliminado	Boolean	Baja lógica
hashPassword	String	NOT NULL, máx. 255
salt	String	máx. 64
ultimoCambio	java.time.LocalDateTime	
requiereReset	Boolean	NOT NULL

---

3) Vehiculo → SeguroVehicular

Clase Vehiculo (A)

Campo	Tipo (Java)	Reglas / Notas
id	Long	PK
eliminado	Boolean	Baja lógica
dominio	String	NOT NULL, **UNIQUE**, máx. 10
marca	String	NOT NULL, máx. 50
modelo	String	NOT NULL, máx. 50
anio	Integer	
nroChasis	String	**UNIQUE**, máx. 50
seguro	SeguroVehicular	Referencia 1→1 a B

Clase SeguroVehicular (B)

Campo	Tipo (Java)	Reglas / Notas
id	Long	PK
eliminado	Boolean	Baja lógica
aseguradora	String	NOT NULL, máx. 80
nroPoliza	String	**UNIQUE**, máx. 50
cobertura	Enum {RC, TERCEROS, TODO_RIESGO}	NOT NULL
vencimiento	java.time.LocalDate	NOT NULL

---

4) DispositivoIoT → ConfiguracionRed

Clase DispositivoIoT (A)

Campo	Tipo (Java)	Reglas / Notas
id	Long	PK
eliminado	Boolean	Baja lógica
serial	String	NOT NULL, **UNIQUE**, máx. 50
modelo	String	NOT NULL, máx. 50
ubicacion	String	máx. 120
firmwareVersion	String	máx. 30
configuracionRed	ConfiguracionRed	Referencia 1→1 a B

Clase ConfiguracionRed (B)

Campo	Tipo (Java)	Reglas / Notas
id	Long	PK
eliminado	Boolean	Baja lógica
ip	String	máx. 45
mascara	String	máx. 45
gateway	String	máx. 45
dnsPrimario	String	máx. 45
dhcpHabilitado	Boolean	NOT NULL

---

5) Paciente → HistoriaClinica

Clase Paciente (A)

Campo	Tipo (Java)	Reglas / Notas
id	Long	PK

Campo	Tipo (Java)	Reglas / Notas
eliminado	Boolean	Baja lógica
nombre	String	NOT NULL, máx. 80
apellido	String	NOT NULL, máx. 80
dni	String	NOT NULL, **UNIQUE**, máx. 15
fechaNacimiento	java.time.LocalDate	
historiaClinica	HistoriaClinica	Referencia 1→1 a B

Clase HistoriaClinica (B)

Campo	Tipo (Java)	Reglas / Notas
id	Long	PK
eliminado	Boolean	Baja lógica
nroHistoria	String	**UNIQUE**, máx. 20
grupoSanguineo	Enum {A+, A-, B+, B-, AB+, AB-, O+, O-}	
antecedentes	String	TEXT
medicacionActual	String	TEXT
observaciones	String	TEXT

---

6) Empresa → DomicilioFiscal

Clase Empresa (A)

Campo	Tipo (Java)	Reglas / Notas
id	Long	PK
eliminado	Boolean	Baja lógica
razonSocial	String	NOT NULL, máx. 120
cuit	String	NOT NULL, **UNIQUE**, máx. 13
actividadPrincipal	String	máx. 80
email	String	máx. 120
domicilioFiscal	DomicilioFiscal	Referencia 1→1 a B

Clase DomicilioFiscal (B)

Campo	Tipo (Java)	Reglas / Notas
id	Long	PK
eliminado	Boolean	Baja lógica
calle	String	NOT NULL, máx. 100
numero	Integer	

Campo	Tipo (Java)	Reglas / Notas
ciudad	String	NOT NULL, máx. 80
provincia	String	NOT NULL, máx. 80
codigoPostal	String	máx. 10
pais	String	NOT NULL, máx. 80

---

7) Libro → FichaBibliografica

Clase Libro (A)

Campo	Tipo (Java)	Reglas / Notas
id	Long	PK
eliminado	Boolean	Baja lógica
titulo	String	NOT NULL, máx. 150
autor	String	NOT NULL, máx. 120
editorial	String	máx. 100
anioEdicion	Integer	
fichaBibliografica	FichaBibliografica	Referencia 1→1 a B

Clase FichaBibliografica (B)

Campo	Tipo (Java)	Reglas / Notas
id	Long	PK
eliminado	Boolean	Baja lógica
isbn	String	**UNIQUE**, máx. 17
clasificacionDewey	String	máx. 20
estanteria	String	máx. 20
idioma	String	máx. 30

---

8) Mascota → Microchip

Clase Mascota (A)

Campo	Tipo (Java)	Reglas / Notas
id	Long	PK
eliminado	Boolean	Baja lógica
nombre	String	NOT NULL, máx. 60
especie	String	NOT NULL, máx. 30

Campo	Tipo (Java)	Reglas / Notas
raza	String	máx. 60
fechaNacimiento	java.time.LocalDate	
duenio	String	NOT NULL, máx. 120
microchip	Microchip	Referencia 1→1 a B

Clase Microchip (B)

Campo	Tipo (Java)	Reglas / Notas
id	Long	PK
eliminado	Boolean	Baja lógica
codigo	String	NOT NULL, **UNIQUE**, máx. 25
fechaImplantacion	java.time.LocalDate	
veterinaria	String	máx. 120
observaciones	String	máx. 255

---

9) Producto → CodigoBarras

Clase Producto (A)

Campo	Tipo (Java)	Reglas / Notas
id	Long	PK
eliminado	Boolean	Baja lógica
nombre	String	NOT NULL, máx. 120
marca	String	máx. 80
categoria	String	máx. 80
precio	double	NOT NULL, escala sugerida (10,2)
peso	Double	opcional, (10,3)
codigoBarras	CodigoBarras	Referencia 1→1 a B

Clase CodigoBarras (B)

Campo	Tipo (Java)	Reglas / Notas
id	Long	PK
eliminado	Boolean	Baja lógica
tipo	Enum {EAN13, EAN8, UPC}	NOT NULL
valor	String	NOT NULL, **UNIQUE**, máx. 20
fechaAsignacion	java.time.LocalDate	

Campo	Tipo (Java)	Reglas / Notas
observaciones	String	máx. 255

---

10) Propiedad → EscrituraNotarial

Clase Propiedad (A)

Campo	Tipo (Java)	Reglas / Notas
id	Long	PK
eliminado	Boolean	Baja lógica
padronCatastral	String	NOT NULL, **UNIQUE**, máx. 30
direccion	String	NOT NULL, máx. 150
superficieM2	java.math.BigDecimal	NOT NULL, (10,2)
destino	Enum {RES, COM}	
antiguedad	Integer	años
escrituraNotarial	EscrituraNotarial	Referencia 1→1 a B

Clase EscrituraNotarial (B)

Campo	Tipo (Java)	Reglas / Notas
id	Long	PK
eliminado	Boolean	Baja lógica
nroEscritura	String	**UNIQUE**, máx. 30
fecha	java.time.LocalDate	NOT NULL
notaria	String	máx. 120
tomo	String	máx. 10
folio	String	máx. 10
observaciones	String	máx. 255

---

11) Pedido → Envio

Clase Pedido (A)

Campo	Tipo (Java)	Reglas / Notas
id	Long	PK
eliminado	Boolean	Baja lógica
numero	String	NOT NULL, **UNIQUE**, máx. 20

Campo	Tipo (Java)	Reglas / Notas
fecha	java.time.LocalDate	NOT NULL
clienteNombre	String	NOT NULL, máx. 120
total	double	NOT NULL, (12,2)
estado	Enum {NUEVO, FACTURADO, ENVIADO}	NOT NULL
envio	Envio	Referencia 1→1 a B

Clase Envio (B)

Campo	Tipo (Java)	Reglas / Notas
id	Long	PK
eliminado	Boolean	Baja lógica
tracking	String	**UNIQUE**, máx. 40
empresa	Enum {ANDREANI, OCA, CORREO_ARG}	
tipo	Enum {ESTANDAR, EXPRES}	
costo	double	(10,2)
fechaDespacho	java.time.LocalDate	
fechaEstimada	java.time.LocalDate	
estado	Enum {EN_PREPARACION, EN_TRANSITO, ENTREGADO}	

Notas generales

- **eliminado**: usarlo para bajas lógicas (ocultar en listados, no borrar físicamente).
- Validar en Service los **campos obligatorios** y **formatos** (email, DNI, ISBN, dominio, CUIT, etc.).