Trabajo Práctico: Propiedad → Escritura Notarial

Introducción y objetivos del TP

Objetivo general. Diseñar e implementar un sistema simple de gestión inmobiliaria que modele el dominio **Propiedad → Escritura Notarial** (relación 1:1) con persistencia en base de datos relacional, operaciones CRUD, validaciones, transacciones y consultas mediante vistas.

Alcance implementado.

- CRUD para **Propiedad** y **EscrituraNotarial** por consola (AppMenu).
- Relación 1:1: una EscrituraNotarial referencia a **Propiedad** con FK **UNIQUE**.
- **Transacción**: alta conjunta Propiedad + Escritura (commit/rollback).
- **Borrado lógico** con flag eliminado en ambas entidades.
- **Vistas** para reporte/búsqueda: v_propiedades_completas y v_busqueda_avanzada, y vistas auxiliares.
- **Manejo de errores** con jerarquía de excepciones propia y clasificador SQL.

Contexto técnico.

- **Lenguaje:** Java (JDK 17+ sugerido).
- **Base de datos:** MariaDB/MySQL (driver org.mariadb.jdbc.Driver).
- **Acceso a datos:** JDBC (PreparedStatement, ResultSet).
- **Capa de ejecución:** Aplicación de consola.

Arquitectura general y patrones utilizados

Arquitectura en capas

- **entities**: POJOs (modelo de dominio) sin lógica de infraestructura.
- **dao**: Data Access Objects. CRUD con JDBC, *mapeo manual* ResultSet → Entity. Lanza SQLException.
- **services**: Lógica de negocio, validaciones, transacciones. Mapea SQLException → AppException con SqlErrorClassifier.
- **exceptions**: Jerarquía de excepciones de aplicación (AppException base; ValidationException, IntegrityException, NotFoundException, DatabaseException).
- **config**: DatabaseConnection (JDBC URL, driver, fail-fast de configuración).
- **main**: AppMenu (UI por consola; orquesta Services y presenta mensajes).

Patrones aplicados.

- **DAO** para separar persistencia.
- **Service Layer** para orquestación/validaciones y transacciones.
- **Factory simple**: DatabaseConnection.getConnection().

Decisiones de diseño más importantes

1) Relación 1:1 y dónde ubicar la FK

- **Qué decidimos.** FK esn_propiedad_id en **escritura_notarial**, con **UNIQUE**, garantizando 1:1.
- **Por qué.** La escritura nace *después* de crear la propiedad; FK en escritura simplifica el flujo de alta transaccional.
- **Alternativas.** PK compartida (mismo id en ambas tablas) o FK en propiedad.
- **Consecuencias.** La escritura depende de una propiedad existente; la unicidad previene asignar una escritura a varias propiedades.

2) Borrado lógico con `eliminado` (boolean)

- **Qué decidimos.** No se borran filas; se marca *_eliminado = TRUE.
- **Por qué.** Trazabilidad, seguridad para TP y pruebas.
- **Alternativas.** Borrado físico; columnas deleted_at/deleted_by
- **Consecuencias.** Todas las consultas/CRUD deben filtrar eliminado = FALSE.

3) Validaciones en Service, no en Entity ni en DAO

- **Qué decidimos.** Reglas de negocio (p.ej., superficie > 0, destino RES/COM, formatos) en **Service**. DAO sólo lo mínimo.
- **Por qué.** Separa responsabilidades y permite reutilizar validación.
- **Alternativas.** Validar en DAO (mezcla capas) o en UI (repetido, inseguro).
- **Consecuencias.** Services tiran ValidationException; DAOs mantienen SQLException puro.

4) Manejo de errores con jerarquía propia

- **Qué decidimos.** Mapear errores SQL a excepciones propias: IntegrityException (duplicados/FK), DatabaseException, etc.
- **Por qué.** Mensajes claros a usuario y control en AppMenu.
- **Alternativas.** Propagar SQLException crudas.
- **Consecuencias.** Services declaran throws AppException; AppMenu captura por tipo y muestra feedback.

5) Transacción para “Propiedad + Escritura”

- **Qué decidimos.** Crear Propiedad y su Escritura en **una sola transacción** (commit/rollback).

- **Por qué.** Consistencia: o se crean ambas, o ninguna.
- **Alternativas.** Dos operaciones sueltas con reintentos (riesgo de inconsistencias).
- **Consecuencias.** El Service controla autoCommit, commit, rollback y reestablecimiento.

6) Vistas para reportes/búsquedas

- **Qué decidimos.** Usar vistas v_propiedades_completas y v_busqueda_avanzada para consultas típicas.
- **Por qué.** Simplifican SQL de consumo y permiten lógica pre-calculada (rangos, texto combinando campos).
- **Alternativas.** Consultas ad-hoc en DAO o lógica en Java.
- **Consecuencias.** Acopla a la BD, pero facilita el TP y mejora legibilidad.

7) Constraint de formato en padrón catastral

- **Qué decidimos.** CHECK con regex tipo ^PC-[0-9]{6}-[0-9]{4}$.
- **Por qué.** Asegura consistencia del dato según consigna/semilla.
- **Alternativas.** Validación sólo en app; regex más flexible por año dinámico.
- **Consecuencias.** Intentos con -ABCD fallan (integridad).

Justificación de tecnologías / lenguajes elegidos

- **Java (JDK 17+)**: alineado al plan de la materia; fuerte tipado; fácil uso de JDBC.
- **MariaDB/MySQL**: disponibilidad local (XAMPP/MariaDB), SQL conocido, soporte de CHECK, ENUM, vistas e índices.
- **JDBC**: transparencia didáctica; control de transacciones y SQL explícito; PreparedStatement mitiga inyección.
- **Consola (AppMenu)**: simplicidad para presentar el TP sin UI gráfica.

Explicación de componentes principales

Paquetes y clases

- **config.DatabaseConnection**: carga driver org.mariadb.jdbc.Driver; provee getConnection(); validación fail-fast de URL/usuario.
- **entities.Propiedad / entities.EscrituraNotarial**: POJOs; Propiedad puede contener referencia a su EscrituraNotarial al listar.
- **dao.PropiedadDao / dao.EscrituraNotarialDao**: CRUD con SQL; versiones con y sin Connection (para transacciones). Lanza SQLException.
- **dao.VistasDao**: consulta vistas (listarPropiedadesCompletas, buscarEnVistaAvanzada, etc.).
- **services.***: validación, orquestación y manejo de transacciones:

- PropiedadService: insertar/actualizar/eliminar/getById/getAll y operación transaccional *alta Propiedad + Escritura*.
- EscrituraNotarialService: CRUD + búsquedas específicas.
- VistasService: fachada limpia de vistas.

- **exceptions.***: jerarquía para manejo de errores en presentación.
- **main.AppMenu**: interfaz por consola; centraliza try/catch por tipo de excepción.

Estructura de carpetas (resumen)

```
src/main/java/
  config/DatabaseConnection.java
  entities/{BaseEntity, Propiedad, EscrituraNotarial, Destino}
  dao/{GenericDao, PropiedadDao, EscrituraNotarialDao, VistasDao}
  services/{GenericService, PropiedadService,
EscrituraNotarialService, VistasService}
  exceptions/{AppException, ValidationException,
IntegrityException, NotFoundException, DatabaseException,
SqlErrorClassifier}
  main/{AppMenu, Main}
```

Desafíos técnicos y cómo los resolvimos

1. **Transacción consistente** en *alta Propiedad + Escritura*.
   - *Riesgo*: crear propiedad sin escritura si falla el segundo insert.
   - *Resolución*: setAutoCommit(false), commit() al final, rollback() en catch, restaurar autoCommit en finally.
2. **Manejo de errores claro en AppMenu.**
   - *Síntoma*: mensajes genéricos.
   - *Resolución*: jerarquía de excepciones y SqlErrorClassifier para mapear SQLException a tipos semánticos.

Alternativas consideradas

- **DAO + JDBC** vs **ORM**: se eligió JDBC por control y didáctica (menos productividad, más código de mapeo).
- **Validación en Service** vs DAO/UI: Service centraliza reglas; DAO queda delgado.
- **Borrado lógico** vs físico: lógico para seguridad; requiere filtros en todas las consultas.
- **Vistas** vs consultas Java: vistas encapsulan complejidad y mejoran legibilidad a costa de mayor dependencia de la BD.

- **Regex de padrón fija** vs dinámica: fija asegura reproductibilidad; dinámica reduce errores de carga pero introduce dependencia temporal.

Decisiones sobre base de datos o persistencia

- **Esquema** (resumen):
  - propiedad: pro_id PK; pro_padron_catastral UNIQUE + CHECK regex; pro_destino enum (RES/COM); pro_eliminado boolean.
  - escritura_notarial: esn_id PK; esn_nro_escritura UNIQUE; esn_propiedad_id FK UNIQUE (1:1); esn_eliminado boolean.
- **Índices**: por UNIQUE y FKs; vistas para lectura/reportes.
- **Vistas**: v_propiedades_completas (join + campos calculados), v_busqueda_avanzada (texto + filtros por rango/año/mes).

API (contratos internos) y manejo de errores

- **DAO**: GenericDao<T> con métodos crear/leer/actualizar/eliminar y variantes transaccionales con Connection. Firma con SQLException.
- **Service**: GenericService<T> con firmas throws AppException. Valida entradas, orquesta transacciones y mapea SQL.
- **Excepciones**:
  - ValidationException: datos inválidos (formatos, nulos requeridos, rangos).
  - IntegrityException: violaciones de unicidad/FK/CHECK.
  - NotFoundException: entidad inexistente (p.ej., getById).
  - DatabaseException: problemas de conexión/IO/driver.
  - SqlErrorClassifier: traduce SQLException a las anteriores.

Consideraciones de seguridad

- **SQL Injection**: todo acceso usa PreparedStatement.
- **Principio de menor privilegio**: sugerido crear un usuario de BD con permisos acotados (lectura/escritura sobre el esquema del TP).
- **Validaciones de entrada**: formato de email, destino, superficies y padrón (también reforzados con CHECK).
- **Errores**: se evita exponer trazas crudas en producción (en el TP se deja comentado el printStackTrace).

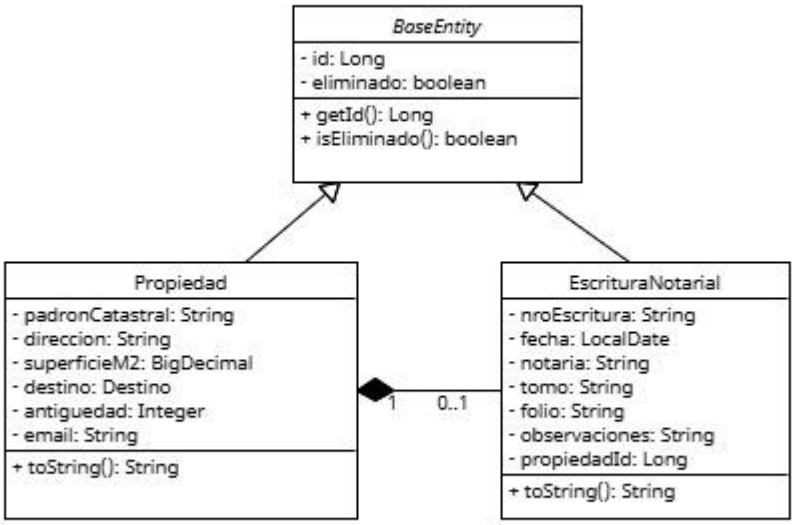
Decisiones sobre testing

- **Pruebas manuales** mediante AppMenu (casos nominales y de error: duplicados, FK inválidas, formato padrón, etc.).

Explicación de flujos principales

1. **Alta Propiedad (simple):** AppMenu → PropiedadService.insertar() → valida → PropiedadDao.crear() → OK/IntegrityException.
2. **Alta Propiedad + Escritura (TX):** AppMenu → PropiedadService.altaPropiedadConEscrituraTransaccional() → setAutoCommit(false) → PropiedadDao.crear() → EscrituraNotarialDao.crear() → commit() o rollback() y excepción mapeada.
3. **Listar Propiedades:** Service obtiene lista y carga escritura asociada (o se apoyan en vistas).
4. **Búsqueda avanzada:** AppMenu pide filtros → VistasService.buscarAvanzado() → VistasDao ejecuta vista con filtros opcionales.

UML



Repositorio

Video