Trabajo Final Integrador (TFI)

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

GRUPO N° 37: (DispositivoIoT – ConfiguracionRed)

- **Elías Carulla** – Desarrollo de la lógica CRUD y conexión JDBC
- **Nicolás González** – Diseño de base de datos y pruebas funcionales
- **Jeremías Wilvers Ocampo** – Documentación, validaciones y manejo de excepciones

LINK AL VIDEO EN YOUTUBE: https://youtu.be/rakAvFnYsR0

1. Introducción.

En este proyecto se trabajó en el desarrollo de una aplicación CRUD (Create, Read, Update, Delete) utilizando Java como lenguaje de programación y MySQL como sistema gestor de base de datos. El objetivo fue comprender cómo interactúan ambos componentes a través de JDBC y cómo implementar operaciones básicas sobre una tabla.

Elección del Dominio y Justificación:

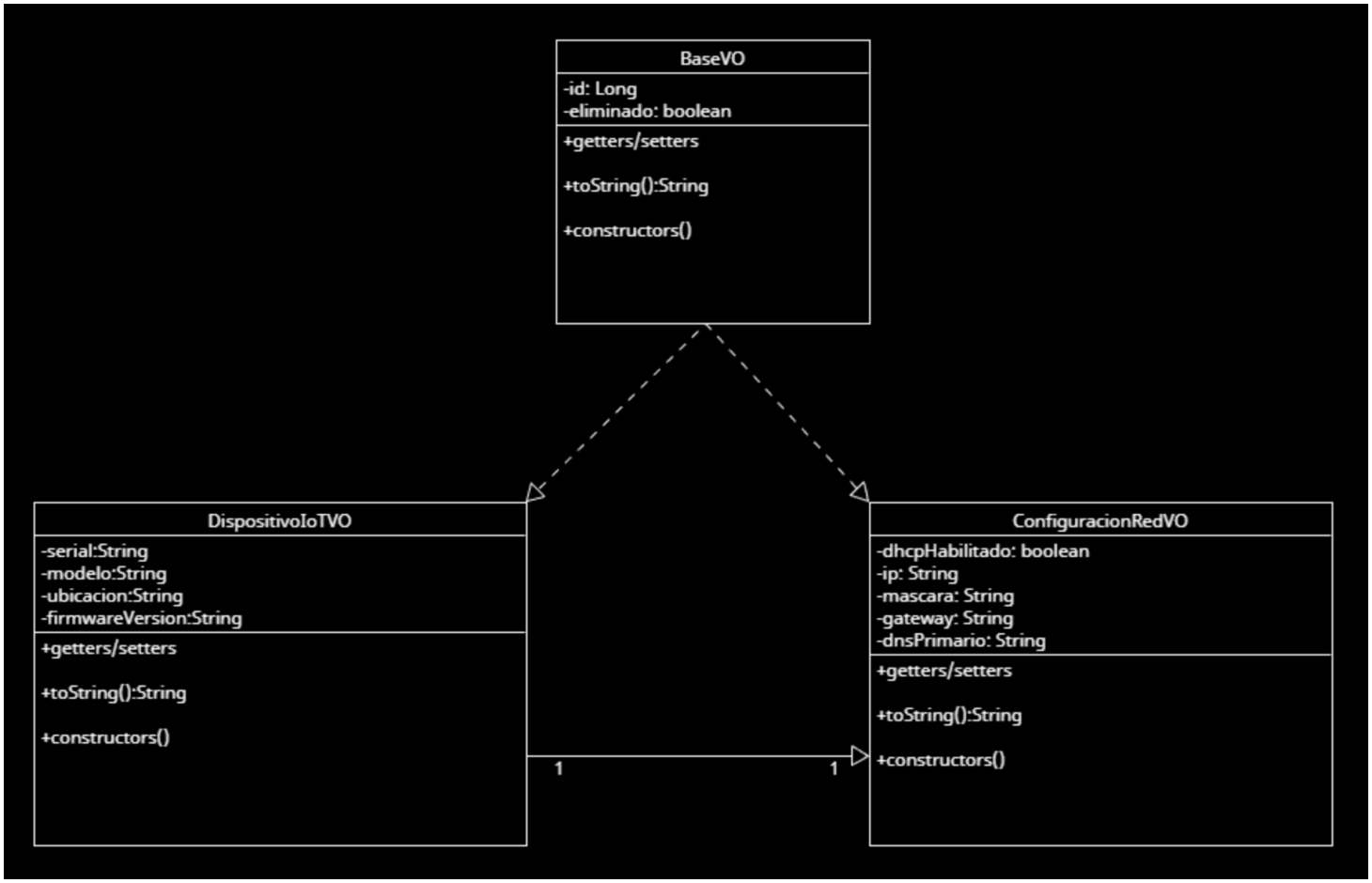
Se eligió el dominio **DispositivoIoT – ConfiguraciónRed** por su relevancia en entornos tecnológicos actuales, donde la gestión de dispositivos conectados y sus configuraciones de red es fundamental. Este modelo permite representar una relación real entre un dispositivo físico y su configuración de red, facilitando la comprensión de relaciones 1→1 en bases de datos y su implementación en Java.

Diseño y Decisiones Clave:

- Se implementó una relación **1→1 unidireccional** entre DispositivoIoT y ConfiguracionRed, donde cada dispositivo tiene una única configuración de red.

- Se optó por una **clave foránea única (FK)** en DispositivoIoT apuntando a ConfiguracionRed, en lugar de una clave primaria compartida, para mantener independencia entre entidades y facilitar pruebas CRUD.

Diagrama UML:



2. Configuración Inicial del Proyecto.

2.1. Entorno de Desarrollo

Se utilizó un entorno Java estándar (JDK), como IDE, NetBeans y como servidor MySQL. También se incorporó el conector JDBC para establecer la comunicación entre Java y la base de datos.

2.2. Creación de la Base de Datos

Se creó una base de datos en MySQL junto con una tabla destinada a almacenar la información de las entidades principales trabajadas.

La estructura incluyó:

Dentro de la entidad **DispositivoIoT**:

- PK idDispositivoIoT INT.
- eliminado BOOLEAN / TINYINT.
- serial VARCHAR(50).
- modelo VARCHAR (50).
- ubicación VARCHAR(120).
- firmawareVersion VARCHAR(30).
- FK idConfiguracion INT.

Dentro de la entidad **ConfiguracionRed**:

- PK ConfiguracionRed INT.
- eliminado BOOLEAN / TINYINT.
- ip VARCHAR(45).
- mascara VARCHAR (45).
- gateway VARCHAR(45).
- dnsPrimario VARCHAR(45).
- dhcpHabilitado VARCHAR(45).

3. Arquitectura por Capas.

El proyecto se estructuró en capas con responsabilidades claras:

- **Modelo (model)**: clases DispositivoIoT y ConfiguracionRed, con atributos y métodos getter/setter.
- **DAO (dao)**: clases encargadas de la persistencia, con métodos CRUD que interactúan con la base de datos mediante JDBC.
- **Conexión (db)**: clase “databaseConection” que gestiona el acceso a MySQL.
- **Vista (main)**: menú en consola para interactuar con el usuario.
- **Controlador (implícito)**: lógica de flujo entre vista y DAO

4. Persistencia: Estructura, Orden y Transacciones.

Estructura de la base

- Dos tablas: DispositivoIoT y ConfiguracionRed, con relación 1→1 mediante FK.
- Campos bien definidos con tipos adecuados (VARCHAR, INT, BOOLEAN).

Orden de operaciones

- Primero se crea la configuración de red.
- Luego se crea el dispositivo, asignándole la configuración.

Transacciones

- La transacción se aplica en un caso donde se quiere crear tanto un dispositivo como una configuración de manera independiente, pero en el mismo momento. Esto se hace mediante el método "crearDispositivoYConfiguracionIndependientes" y utilizando métodos de la clase TransactionManager tales como "rollback" y "startTransaction". Lo que hace la transacción es crear un dispositivo y una configuración, en el caso de que alguna de las dos falle, no se realizan cambios a la base de datos y el rollback se ejecuta.

5. Validaciones y Reglas de Negocio.

Validación de campos vacíos o inválidos antes de insertar.

- Verificación de existencia de ID antes de actualizar o eliminar.
- Uso de “PreparedStatement” para evitar inyección SQL.
- Mensajes claros al usuario según el resultado de cada operación.

5. Manejo de Excepciones y Buenas Prácticas.

Durante el desarrollo se incorporaron prácticas como:

- **Controlar excepciones mediante bloques try-catch.**

  Cuando escribís código que interactúa con la base de datos, muchas cosas pueden salir mal.

Por ejemplo:

- MySQL no está en ejecución.
- Error en el usuario o contraseña.
- Consulta SQL mal escrita.
- Error con el tipo de dato enviado.
- Fallas de red o bloqueo en la tabla.

¿Por qué es importante?

- Evita que la aplicación “crashee”.
- Permite mostrar mensajes claros al usuario.
- Hace más fácil encontrar y corregir errores.

- **Cerrar conexiones, sentencias y resultados en bloques finally.**

  Las conexiones JDBC (Connection, PreparedStatement y ResultSet) consumen recursos del sistema y **no se cierran automáticamente**.

  Si no las cerrás:

  - Se acumulan conexiones abiertas.
  - MySQL puede rechazar nuevas conexiones.
  - La aplicación puede volverse lenta o bloquearse.

  ¿Por qué en finally?

  Porque:

  - Si el código funciona → se cierra igual.
  - Si el código falla → también se cierra.
  - Garantiza que siempre se liberen los recursos.

- **Mostrar mensajes informativos sobre el éxito o falla de cada operación.**

Esto significa que la aplicación no debe trabajar “en silencio”. El usuario necesita saber qué pasó con la operación que intentó realizar.

Por Ejemplo:

Si se inserta un dato correctamente:

System.out.println("Dispositivo registrado con éxito.");

Si falla la inserción:

System.out.println("No se pudo registrar el Dispositivo. Verifique que todo sea correcto.");

Si se intenta actualizar un ID inexistente:

System.out.println("No existe un Dispositivo con ese ID.");

¿Por qué es útil esto?

- Mejora la experiencia del usuario.
- Facilita la depuración durante el desarrollo.
- Permite validar que las reglas de negocio están funcionando.

6. Pruebas del Funcionamiento.

Se realizaron pruebas para cada operación CRUD:

- **Create**: inserción de dispositivos y configuraciones, verificación en MySQL.
- **Read**: listado completo y búsqueda por ID, validación de mapeo correcto.
- **Update**: modificación de registros existentes, pruebas con IDs inválidos.
- **Delete**: eliminación por ID, confirmación en base de datos.

Inserción de nuevos datos:

Se probó la funcionalidad de agregar registros a la base de datos utilizando métodos INSERT. Esto implicó:

✓ Crear nuevos DispositivosIoT y ConfiguracionRed desde el menú.

- ✓ Verificar que los datos se almacenaran correctamente en MySQL.
- ✓ Confirmar el correcto uso de parámetros en PreparedStatement para evitar inyección SQL.

Lectura de la tabla completa y búsqueda por ID

- ✓ Listar todos los registros existentes, comprobando su visualización en consola.
- ✓ Realizar búsquedas por ID, verificando que la aplicación respondiera adecuadamente tanto si el registro existía como si no.
- ✓ Confirmar la correcta interpretación de los datos obtenidos desde ResultSet.

Modificación de campos específicos

Se probaron las operaciones UPDATE verificando:

- ✓ La correcta actualización de un registro ya existente.
- ✓ El uso adecuado de parámetros indexados dentro del PreparedStatement.
- ✓ Validaciones antes de permitir la modificación.
- ✓ Comportamiento ante IDs inexistentes.

Eliminación de registros

Las pruebas DELETE consistieron en:

- ✓ Eliminar un registro mediante su ID.
- ✓ Verificar que la eliminación fuera permanente en la tabla.

Estas pruebas aseguraron la estabilidad del sistema y el correcto manejo de la integridad de la base.

7. Conclusión.

El proyecto permitió comprender el ciclo completo de interacción entre una aplicación Java y una base de datos MySQL. Se aplicaron conceptos fundamentales como conexiones JDBC, consultas parametrizadas, manejo de excepciones y estructuración de un CRUD básico. Este desarrollo sentó las bases para futuros proyectos más complejos, tanto en programación como en diseño de aplicaciones en los cuatrimestres posteriores de carrera.

8. Fuentes y Herramientas Utilizadas.

- **Lenguaje**: Java (JDK 21)

- **IDE**: NetBeans y Eclipse
- **Base de datos**: MySQL
- **Conector**: JDBC