Trabajo Final Integrador

Materia: Programacion 2

Dominio elegido: Empresa - Domicilio Fiscal (relacion 1 a 1)

Link al video:
https://drive.google.com/file/d/1oS9Nx22xFaEM4fFwVojaLOfTW6xKJxe6/view?usp=sharing

Integrantes:

Lorenzo Blanco, Rol: Entities y Config

Tiziano Caamaño, Rol: Main (AppMenu) y SQL

Brian David Apaza Gamarra, Rol: Service

Kenyi Alejandro Meza Andagua, Rol: DAO

1. Introduccion

En el presente trabajo final integrador se desarrollo una aplicacion en Java que implementa una relacion unidireccional 1 a 1 entre las clases Empresa y DomicilioFiscal. El objetivo principal fue aplicar los conceptos vistos en la materia Programacion 2: programacion orientada a objetos, acceso a bases de datos mediante JDBC, patron DAO, capa de servicios con manejo de transacciones y construccion de un menu de consola para realizar operaciones CRUD.

La idea fue simular un pequeno sistema de gestion de empresas donde, para cada empresa, se puede registrar su domicilio fiscal asociado. A partir de este requerimiento se diseniaron las entidades, la base de datos en MySQL y la arquitectura por capas del proyecto.

2. Dominio elegido y justificacion

Para el TFI se eligio el dominio Empresa - DomicilioFiscal. Este dominio nos parecio adecuado porque es sencillo de entender pero al mismo tiempo permite reflejar claramente la relacion 1 a 1 planteada en el enunciado: una empresa puede tener un unico domicilio fiscal asociado.

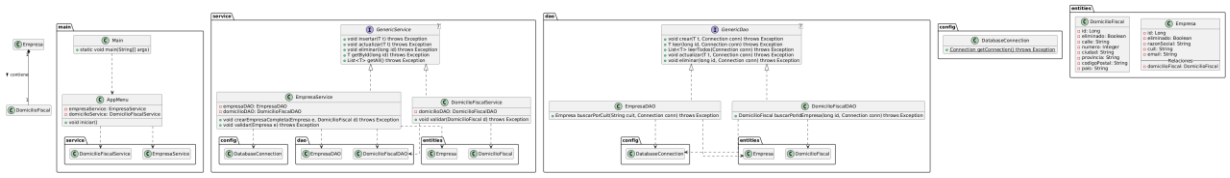
Ademas, se trata de un caso muy comun en sistemas reales de gestion administrativa y contable, lo cual ayuda a visualizar como se podria extender el modelo en el futuro (por ejemplo, incorporando empleados, sucursales u otros tipos de domicilios).

3. Diseño y modelado de clases

La clase Empresa (A) representa a la entidad principal del sistema. Contiene los campos id, eliminado, razonSocial, cuit, actividadPrincipal, email y la referencia domicilioFiscal de tipo DomicilioFiscal. La clase DomicilioFiscal (B) incluye id, eliminado, calle, numero, ciudad, provincia, codigoPostal y pais.

Se implementaron constructores por defecto y constructores con parametros, junto con los correspondientes getters y setters. Tambien se sobreescribio el metodo toString() en ambas clases para facilitar la visualizacion de los objetos en la consola. En el caso de Empresa, el toString evita imprimir recursivamente el domicilio completo y solo indica si la empresa tiene o no un domicilio asociado.

3.1. Diagrama UML



4. Base de datos MySQL y relacion 1 a 1

La base de datos se creo en MySQL utilizando el script 01_create_database.sql. En dicho script se define la base tpi_empresas y las tablas empresa y domicilio_fiscal.

La tabla empresa contiene las columnas id (BIGINT, PK, autoincremental), eliminado (BOOLEAN), razon_social, cuit (con restriccion UNIQUE), actividad_principal y email.

La tabla domicilio_fiscal incluye id, eliminado, calle, numero, ciudad, provincia, codigo_postal, pais y la columna empresa_id. Esta ultima es una clave foranea hacia empresa(id) con restriccion UNIQUE, lo que garantiza a nivel de base de datos que cada empresa puede estar asociada a lo sumo a un unico domicilio fiscal. Ademas, la FK esta definida con ON DELETE CASCADE y ON UPDATE CASCADE para mantener la integridad referencial.

5. Arquitectura por capas (DAO, Service y AppMenu)

El proyecto se organizo en los paquetes config, entities, dao, service y main.

En config se encuentra la clase DatabaseConnection, que lee las propiedades de conexion desde un archivo externo db.properties y expone un metodo estatico getConnection() que retorna un objeto java.sql.Connection configurado.

En el paquete dao se definio la interfaz generica GenericDao<T> con los metodos crear, leer, leerTodos, actualizar y eliminar. A partir de esta interfaz se implementaron EmpresaDao y DomicilioFiscalDao, los cuales utilizan PreparedStatement en todas las operaciones y reciben una Connection externa para poder participar de las mismas transacciones.

El paquete service contiene la interfaz GenericService<T> y las clases EmpresaService y DomicilioFiscalService. En esta capa se centralizan las reglas de negocio, las validaciones de campos obligatorios y el manejo de transacciones (commit y rollback).

Finalmente, en main se ubica la clase AppMenu, encargada de interactuar con el usuario a traves de la consola. Desde este menu se invocan los metodos de la capa de servicios para realizar las operaciones CRUD sobre Empresa y DomicilioFiscal.

6. Persistencia y manejo de transacciones

La persistencia se realiza mediante JDBC puro, sin utilizar ningun ORM. La clase DatabaseConnection encapsula la creacion de la conexion hacia MySQL.

En los servicios se aplica el manejo transaccional solicitado en la consigna. Por ejemplo, en EmpresaService.insertar() se obtiene una Connection, se desactiva el autocommit (setAutoCommit(false)) y se llama a EmpresaDao.crear() para insertar la empresa. Si la entidad tiene un DomicilioFiscal asociado, tambien se invoca a DomicilioFiscalDao.crear() y luego a DomicilioFiscalDao.establecerRelacionEmpresa() para guardar la relacion 1 a 1. Solo si todas las operaciones terminan correctamente se hace commit(). En caso de que ocurra alguna excepcion, se realiza rollback() y se propaga el error hacia la capa superior.

Un esquema similar se utiliza en los metodos actualizar() y eliminar() de ambas clases de servicio, garantizando que las operaciones compuestas se ejecuten de manera atomica.

7. Validaciones y reglas de negocio

En EmpresaService se implementaron varias validaciones antes de realizar las operaciones sobre la base de datos. Entre ellas se verifica que la razon social no sea nula ni vacia y que no supere los 120 caracteres, que el CUIT sea obligatorio y no exceda los 13 caracteres y que la actividadPrincipal y el email respeten las longitudes maximas definidas en la especificacion. Tambien se realiza una validacion basica del formato de email comprobando que contenga el caracter "@".

Ademas, se controla la unicidad del CUIT. Antes de insertar una empresa nueva se consulta si ya existe otra empresa con el mismo CUIT y, en caso afirmativo, se lanza una IllegalArgumentException. Durante la actualizacion tambien se evita que dos empresas terminen compartiendo el mismo CUIT.

En DomicilioFiscalService se validan los campos calle, numero, ciudad, provincia, codigoPostal y pais, asegurando que se cumplan las restricciones de obligatoriedad y longitud maxima indicadas en la consigna. Cualquier violacion genera una IllegalArgumentException con un mensaje descriptivo para el usuario.

8. Menu de consola y manejo de errores

La clase AppMenu ofrece un menu principal desde el cual se puede acceder a la gestion de empresas y a la gestion de domicilios fiscales. Cada submenu permite crear, buscar por ID, realizar busquedas por campos clave (como el CUIT o la razon social en el caso de Empresa), listar todos los registros, actualizar y realizar bajas logicas.

Se utilizan estructuras while y switch para manejar las opciones ingresadas por el usuario, convirtiendo la entrada a mayusculas o eliminando espacios cuando corresponde. Tambien se incluyen bloques try/catch para capturar NumberFormatException al parsear IDs numericos y SQLException o IllegalArgumentException provenientes de la capa de servicios. Los mensajes mostrados en consola intentan ser claros, indicando tanto los casos de exito como las posibles causas de error (por ejemplo, ID inexistente, formato invalido o violacion de unicidad).

9. Pruebas realizadas

Para comprobar el correcto funcionamiento del sistema se realizaron distintas pruebas manuales desde el menu de consola. Entre ellas, se crearon varias empresas con y sin domicilio fiscal, se verifico que no sea posible registrar dos empresas con el mismo CUIT y se probaron las operaciones de actualizacion y baja logica.

Tambien se ejecutaron consultas directas en MySQL para confirmar que la relacion 1 a 1 entre empresa y domicilio_fiscal se mantiene correctamente y que el campo eliminado se utiliza de forma consistente en los listados.

10. Conclusiones y trabajos futuros

Como conclusion general, el desarrollo de este TFI permitio integrar varios temas abordados en la materia, desde el disenio orientado a objetos y el modelado de bases de datos hasta el uso de JDBC con transacciones y el patron DAO. La separacion en capas ayudo a mantener el codigo mas organizado y a entender mejor la responsabilidad de cada componente.

Como posibles mejoras futuras se podria agregar una interfaz grafica o una API REST por encima de la logica ya implementada, reutilizando la capa de servicios. Tambien seria

interesante incorporar nuevas entidades relacionadas con la empresa (por ejemplo, empleados o sucursales) y extender el sistema de validaciones.

11. Fuentes y herramientas utilizadas

Para la realizacion de este trabajo se utilizaron, ademas de las clases teoricas y practicas de la materia, la documentacion oficial de Java y MySQL, asi como material de referencia sobre JDBC y el patron DAO. Tambien se emplearon herramientas de apoyo como el IDE IntelliJ IDEA o Eclipse, el sistema de control de versiones Git y GitHub para el repositorio del proyecto.

Se utilizaron asistentes de software como apoyo puntual para revisar fragmentos de codigo o proponer mejoras de redaccion, pero todas las decisiones de disenio y la implementacion final del proyecto fueron analizadas y ajustadas por el equipo para que el resultado sea coherente con lo pedido en la consigna.