Programación 3

Arquitectura de Software (back-end en Java)

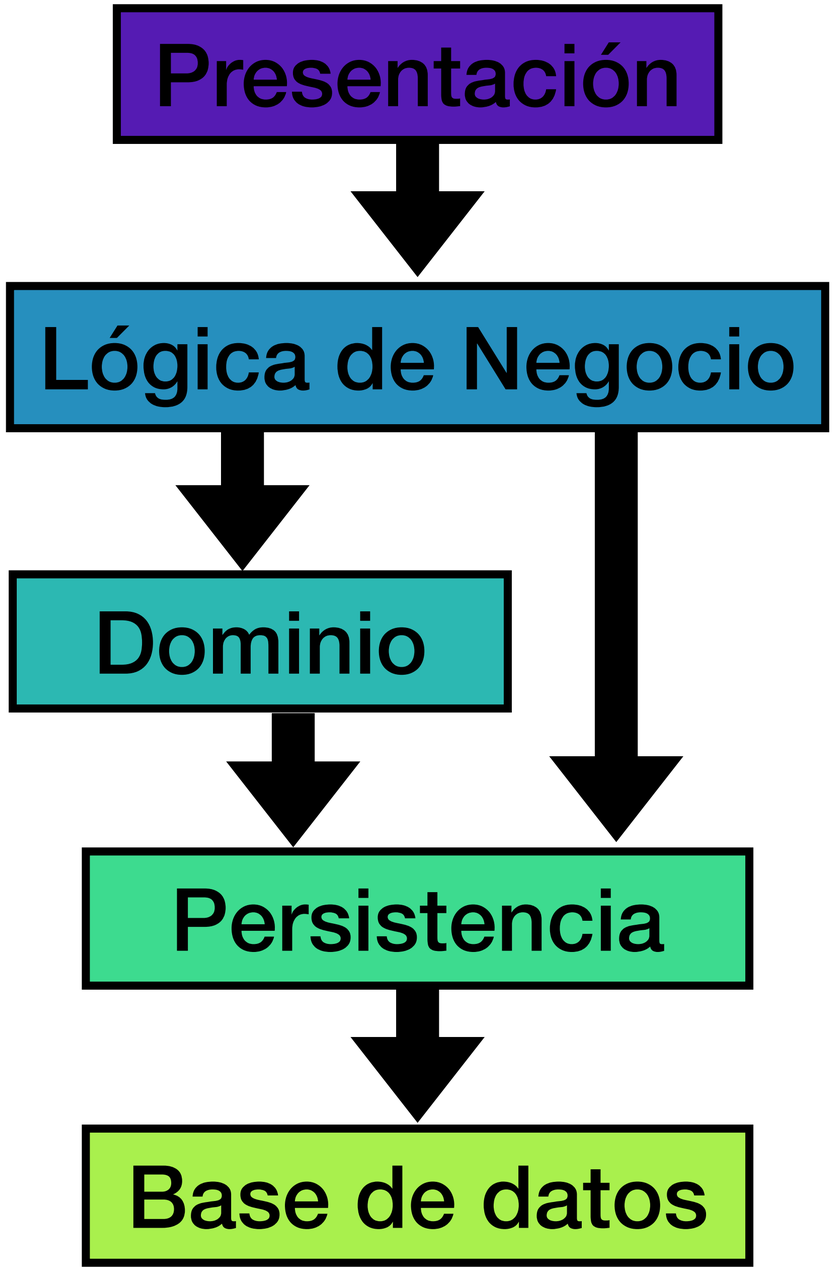
Dr. Andrés Melgar

Arquitectura de Software

Arquitectura de SW
Especificación

- Sigue un estilo arquitectónico monolítico por capas:
  - **Presentación**: Interfaz con la que interactúa el usuario, gestiona la entrada y salida de datos.
  - **Lógica del negocio**: Procesa las reglas y operaciones de la aplicación, coordinando la interacción entre capas.
  - **Dominio**: Representa los objetos y comportamientos del modelo de negocio, independiente de la tecnología.
  - **Persistencia**: Gestiona el acceso y almacenamiento de datos, conectando la aplicación con la base de datos.
  - **Base de datos**: Almacén estructurado donde se guardan y organizan los datos de la aplicación.

Mayor detalle sobre arquitectura de software, su representación y estilos, los verán en el curso 1INF31 - Arquitectura de Software



Arquitectura de SW (estilo arquitectónico monolítico por capas)

Presentación

Front-End

Microsoft .NET

Lógica de Negocio

Java

BussinessObject
+regla_de_negocio()

Patrón DAO
(patrón de interacción con BD Data Mapper)

Dominio

Back-End

Model
-campo1
-campo2

Patrones:
- Data Transfer Object (diseño)
- Identity Field (persistencia)
- Sigle Table Inherence (persistencia)

Persistencia

DataAccessObject
+insertar()
+actualizar()
+eliminar()
+consultar()

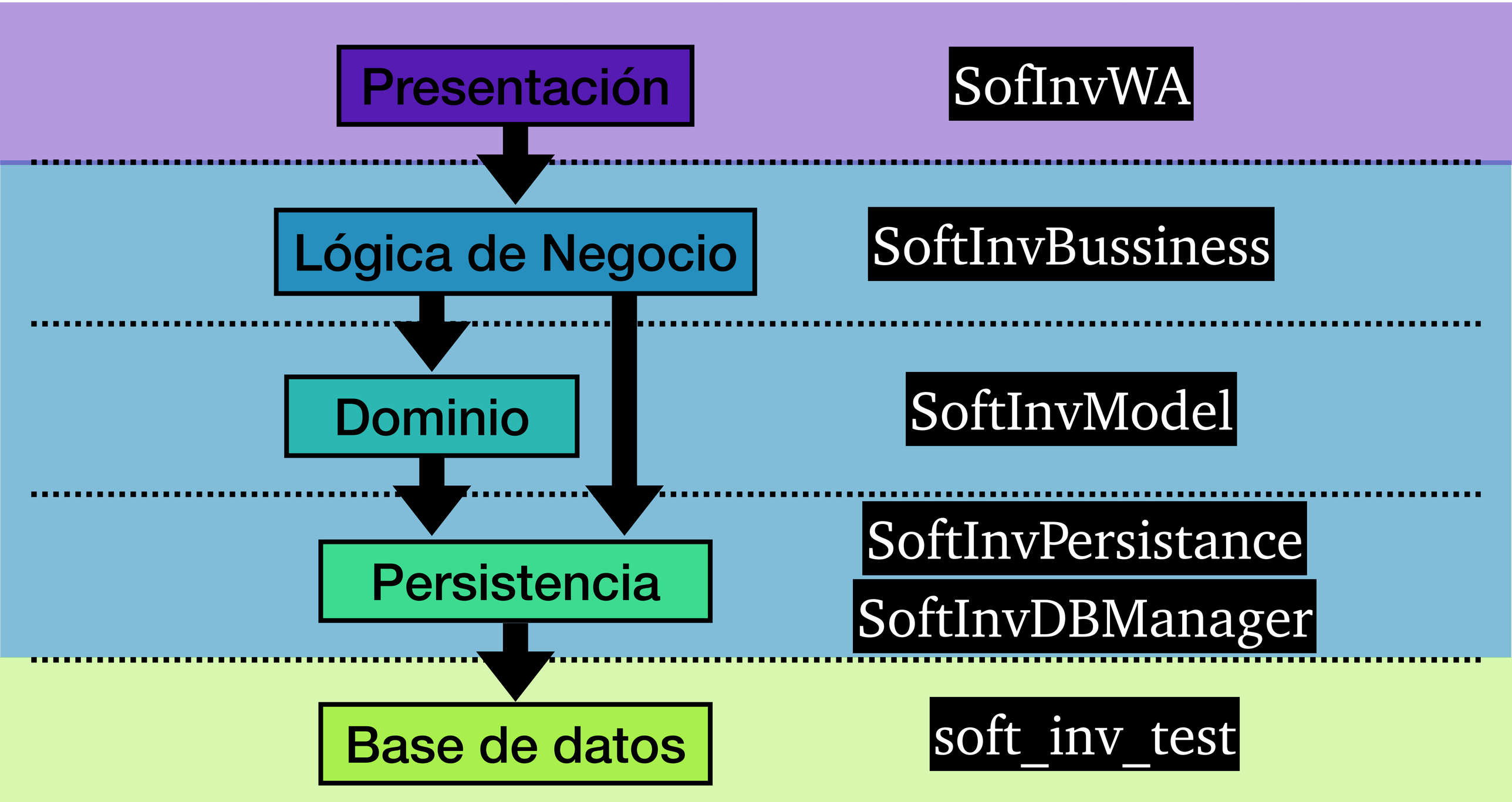
DataAccessObjetImplementation
-conexion
-resultSet
+insertar()
+actualizar()
+eliminar()
+consultar()

Base de datos

MySQL

Microsoft® SQL Server®

Arquitectura de SW (estilo arquitectónico monolítico por capas)



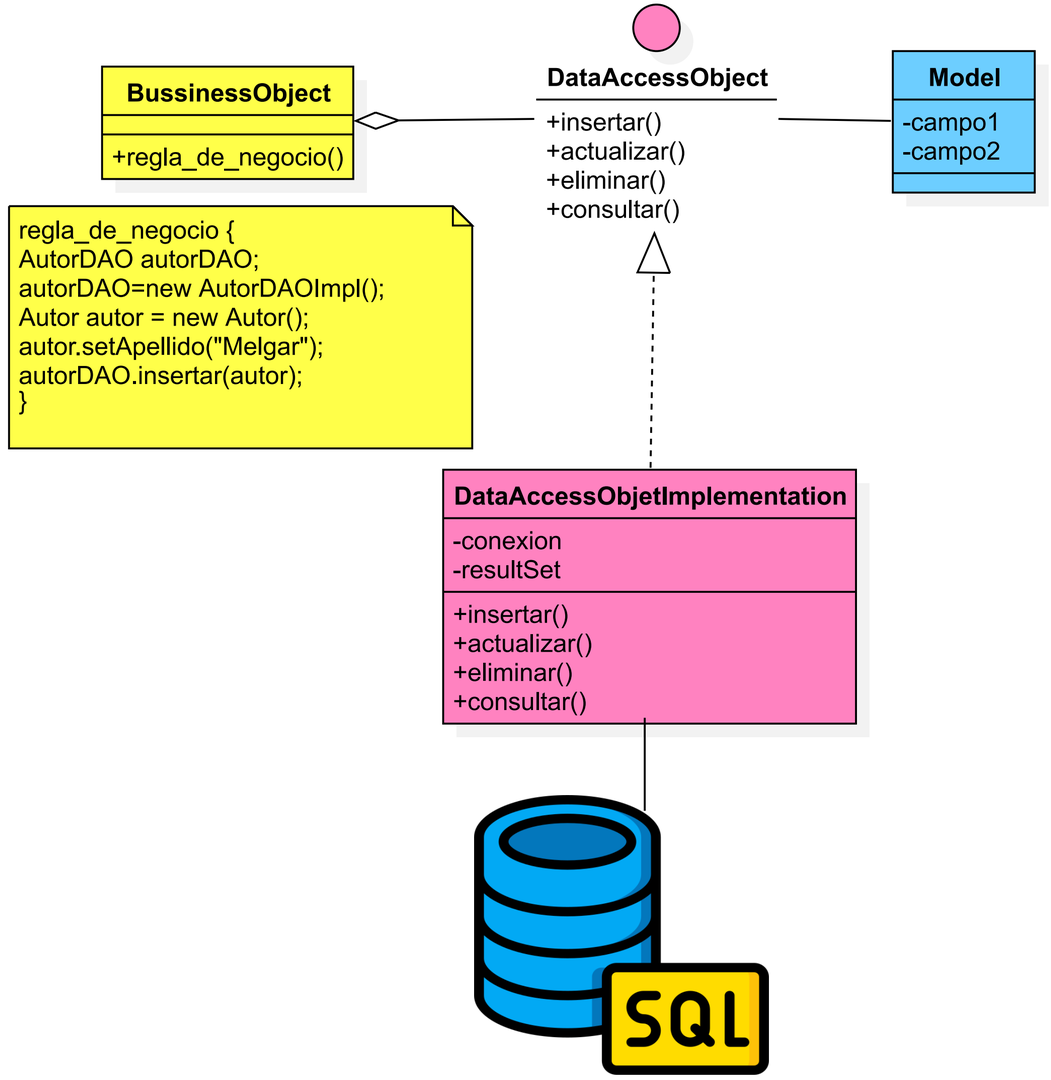
Patrón DAO

El Patrón DAO

Especificación

- El **patrón DAO** (*Data Access Object*) es un patrón de diseño estructural que proporciona una **abstracción** entre la **lógica de negocio** y la forma en que se accede a **los datos** almacenados en una base de datos u otro sistema de persistencia.

- Este patrón permite que la lógica de negocio no dependa directamente de los detalles de acceso a datos, lo que facilita la modificación, mantenimiento y prueba del código.

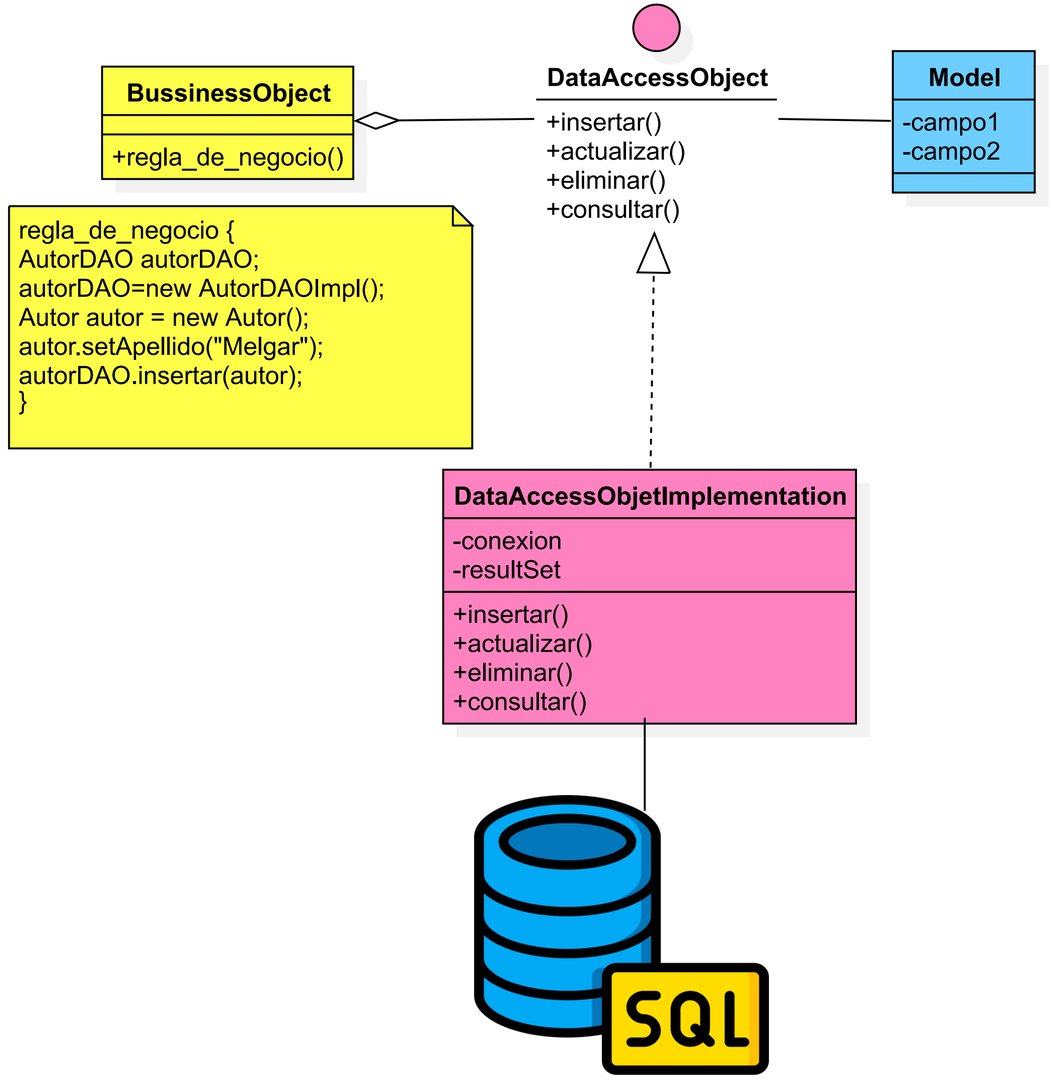


Mayor detalle sobre diagrama de clases de diseño, notación UML y patrones, los verán en el curso 1INF50 - Diseño de Software

El Patrón DAO

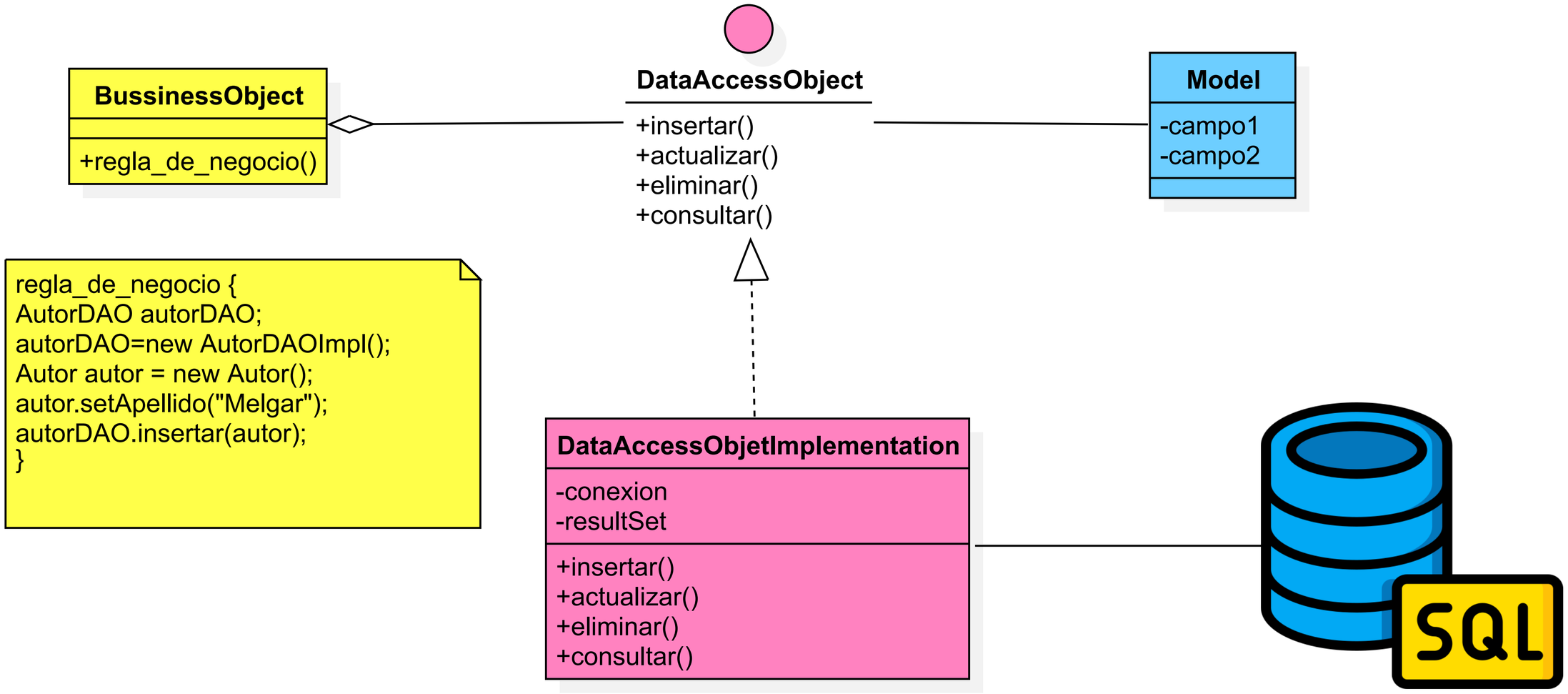
Ventajas

- El **patrón DAO** ofrece las siguientes ventajas:
  - **Desacoplamiento**: Separa la lógica de acceso a datos de la lógica de negocio.
  - **Mantenibilidad**: Cambios en la base de datos no afectan otras partes del sistema.
  - **Reutilización**: Permite reutilizar la lógica de acceso a datos en distintas partes de la aplicación.
  - **Portabilidad**: Facilita cambiar la tecnología de persistencia (JDBC, Hibernate, JPA).



Mayor detalle sobre diagrama de clases de diseño, notación UML y patrones, los verán en el curso 1INF50 - Diseño de Software

El Patrón DAO



Mayor detalle sobre diagrama de clases de diseño, notación UML y patrones, los verán en el curso 1INF50 - Diseño de Software

Capa de Dominio

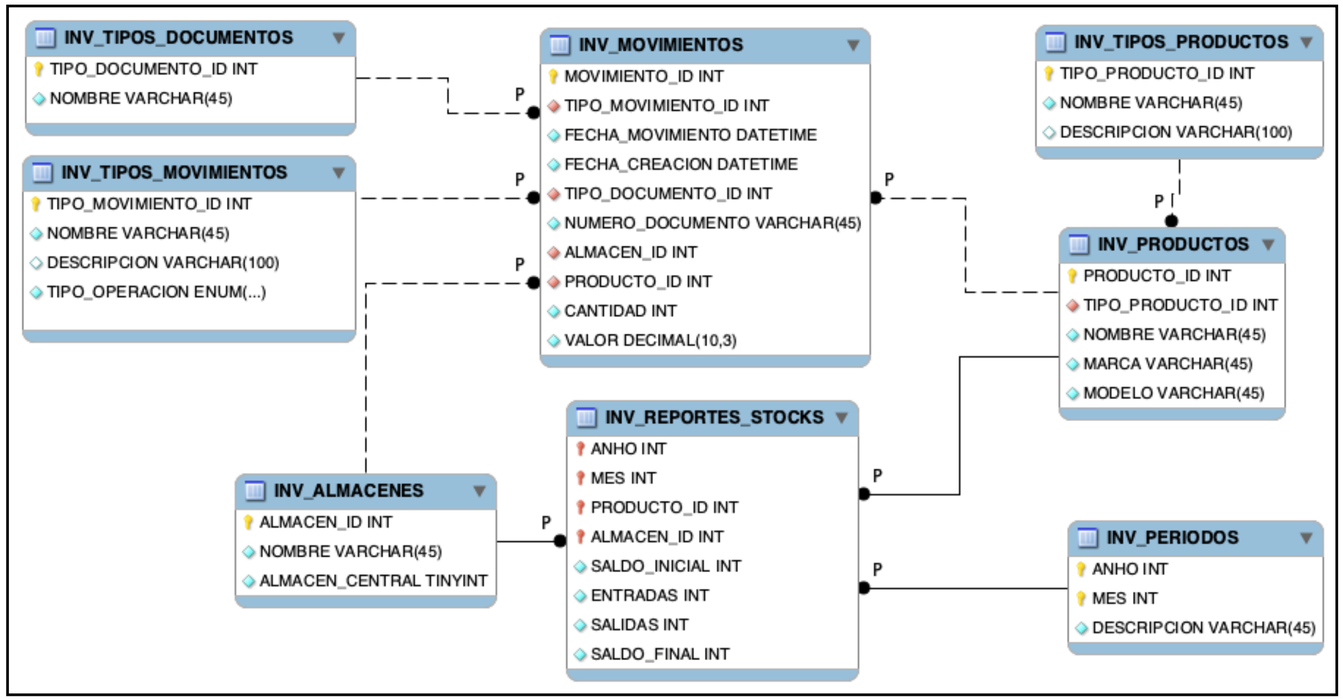
Capa de Dominio

Especificación

- En la capa de dominio se representan los objetos y comportamientos del **modelo de negocio**, independiente de la tecnología.

- Contiene la representación de las de modelo que se almacena en la **base de datos**.

- Se implementará usando el **patrón DTO** (*Data Transfer Object*).



Mayor detalle sobre diagrama de clases de diseño, notación UML y patrones, los verán en el curso 1INF50 - Diseño de Software

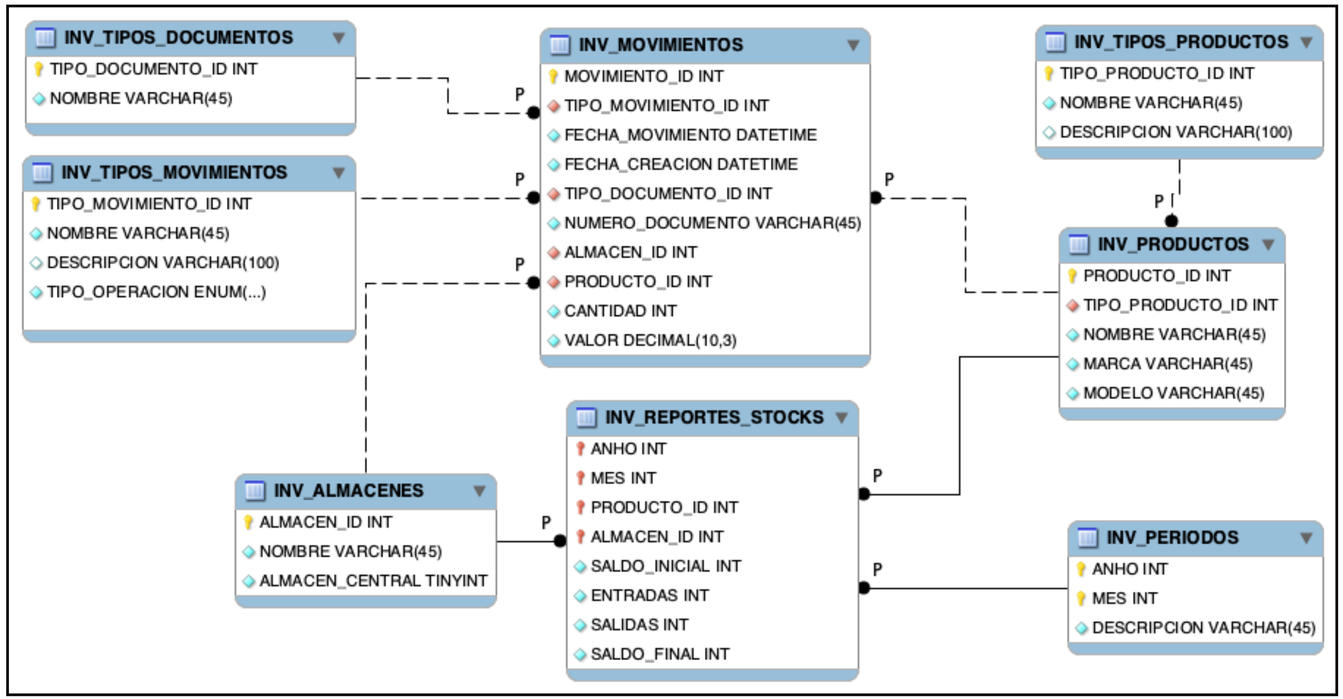
Capa de Dominio

Patrón DTO

- El **patrón DTO** (*Data Transfer Object*) es un patrón de diseño que se usa para **transferir datos** entre capas de una aplicación.

- Su principal objetivo es **transportar información** sin exponer la lógica de negocio ni la estructura interna de la base de datos.

- Un DTO **no tiene lógica de negocio**; solo contiene atributos y métodos getter y setter para acceder a los datos.

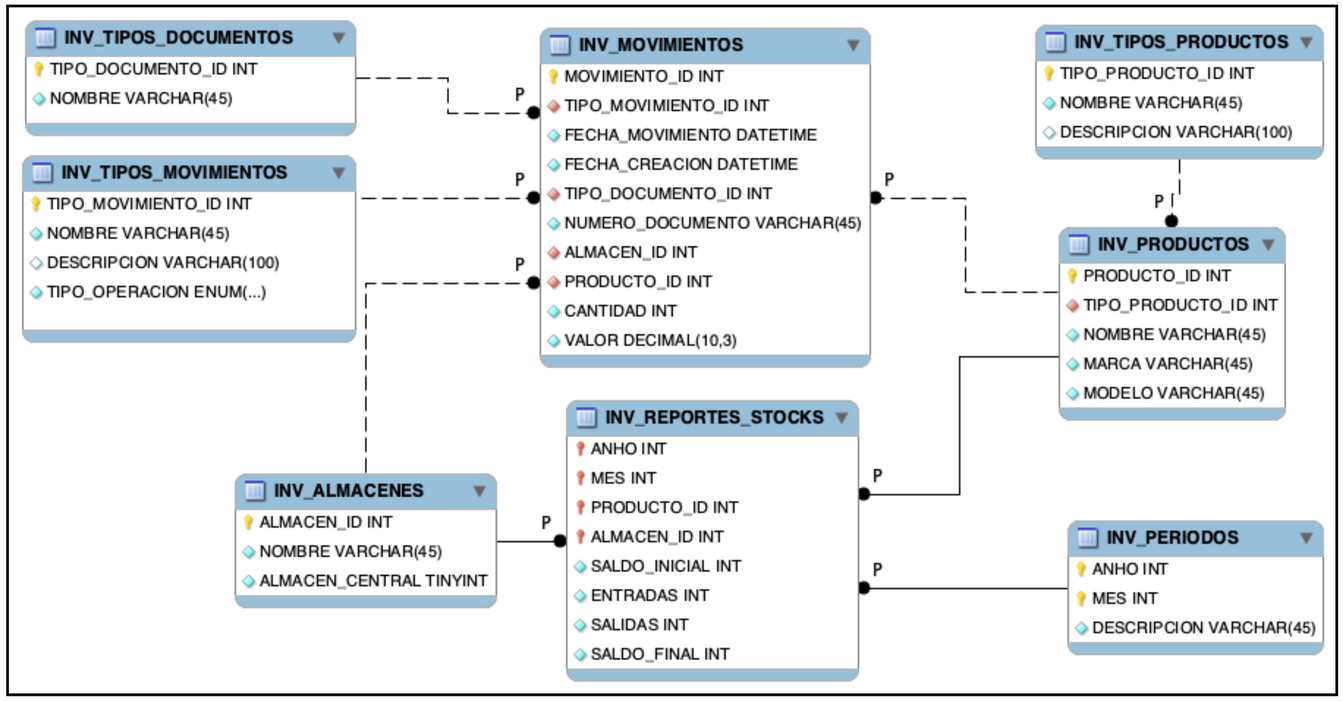


Mayor detalle sobre diagrama de clases de diseño, notación UML y patrones, los verán en el curso 1INF50 - Diseño de Software

Capa de Dominio

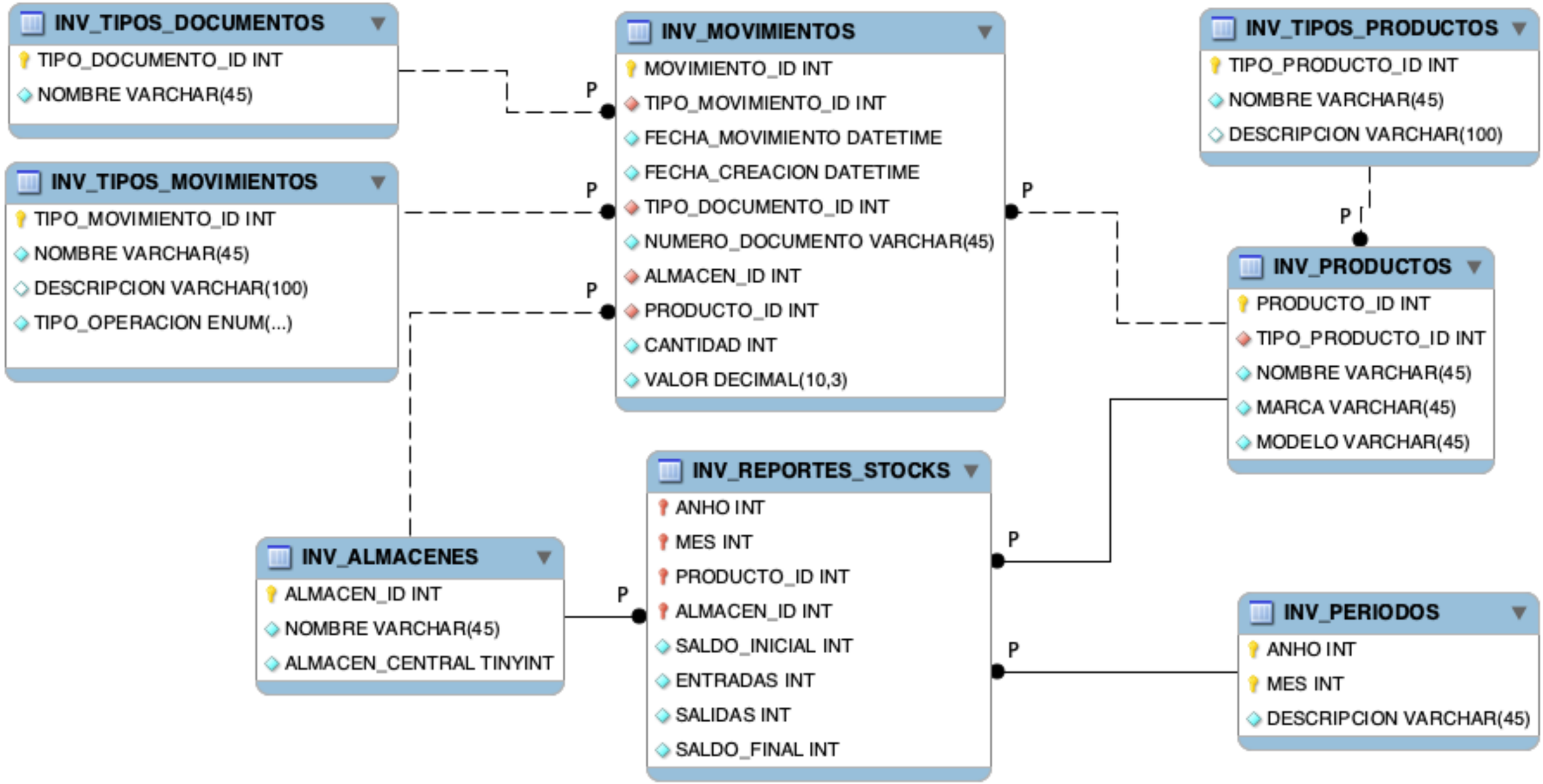
Consideraciones

- Cada tabla se representará como una clase siguiendo el **patrón DTO**.
- Las **llaves primarias** se almacenarán en la misma clase (patrón Identity Field).
- Cuando se tenga una relación entre dos tablas, en lugar de almacenar la **llave foránea**, se almacena la clase completa.
- Si hubiera una **jerarquía de tablas** se mapea en una única clase (patrón Single Table Inherence).

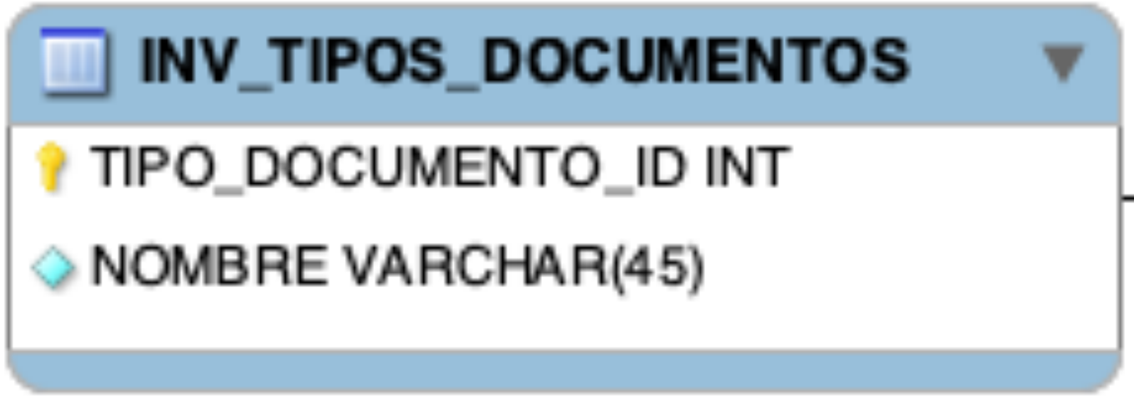


Mayor detalle sobre diagrama de clases de diseño, notación UML y patrones, los verán en el curso 1INF50 - Diseño de Software

Modelo Relacional

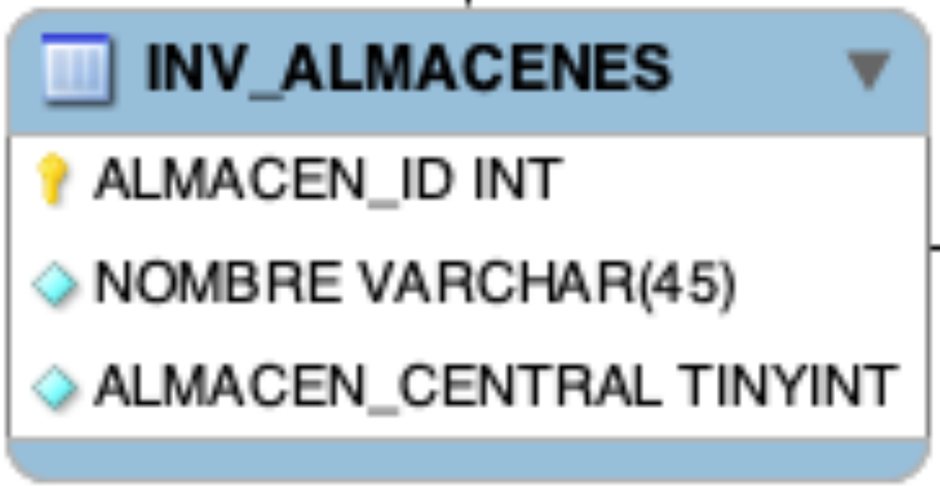


Mapeamiento de tablas y clases



TiposDocumentosDTO

-tipoDocumentoId: Integer
-nombre: String

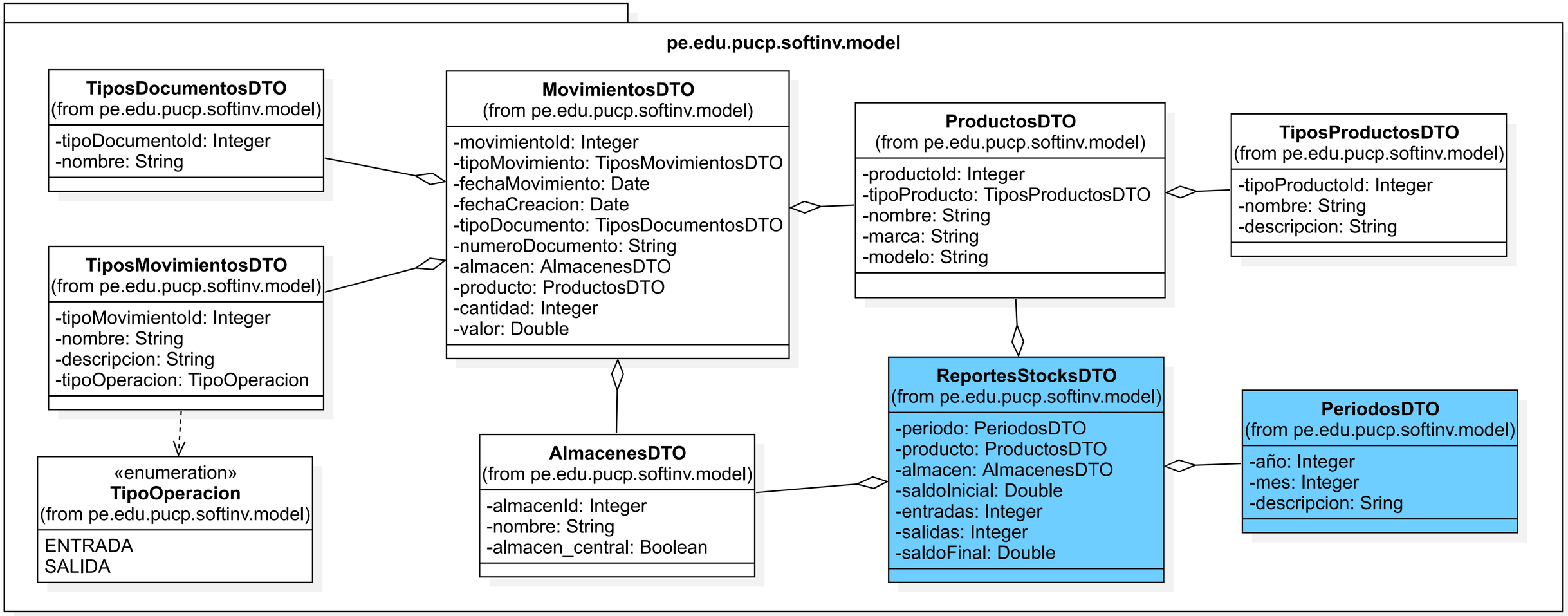


AlmacenesDTO

-almacenId: Integer
-nombre: String
-almacen_central: Boolean

Mayor detalle sobre diagrama de clases de diseño, notación UML y patrones, los verán en el curso 1INF50 - Diseño de Software

Diagrama de clases



Mayor detalle sobre diagrama de clases de diseño, notación UML y patrones, los verán en el curso 1INF50 - Diseño de Software

Capa de Persistencia

Recordando

Arquitectura JDBC

Aplicación en Java

API JDBC

Gestor de drivers JDBC

MySQL driver

ResultSet

ResultSet

ResultSet

Statement

PreparedStatement

CallableStatement

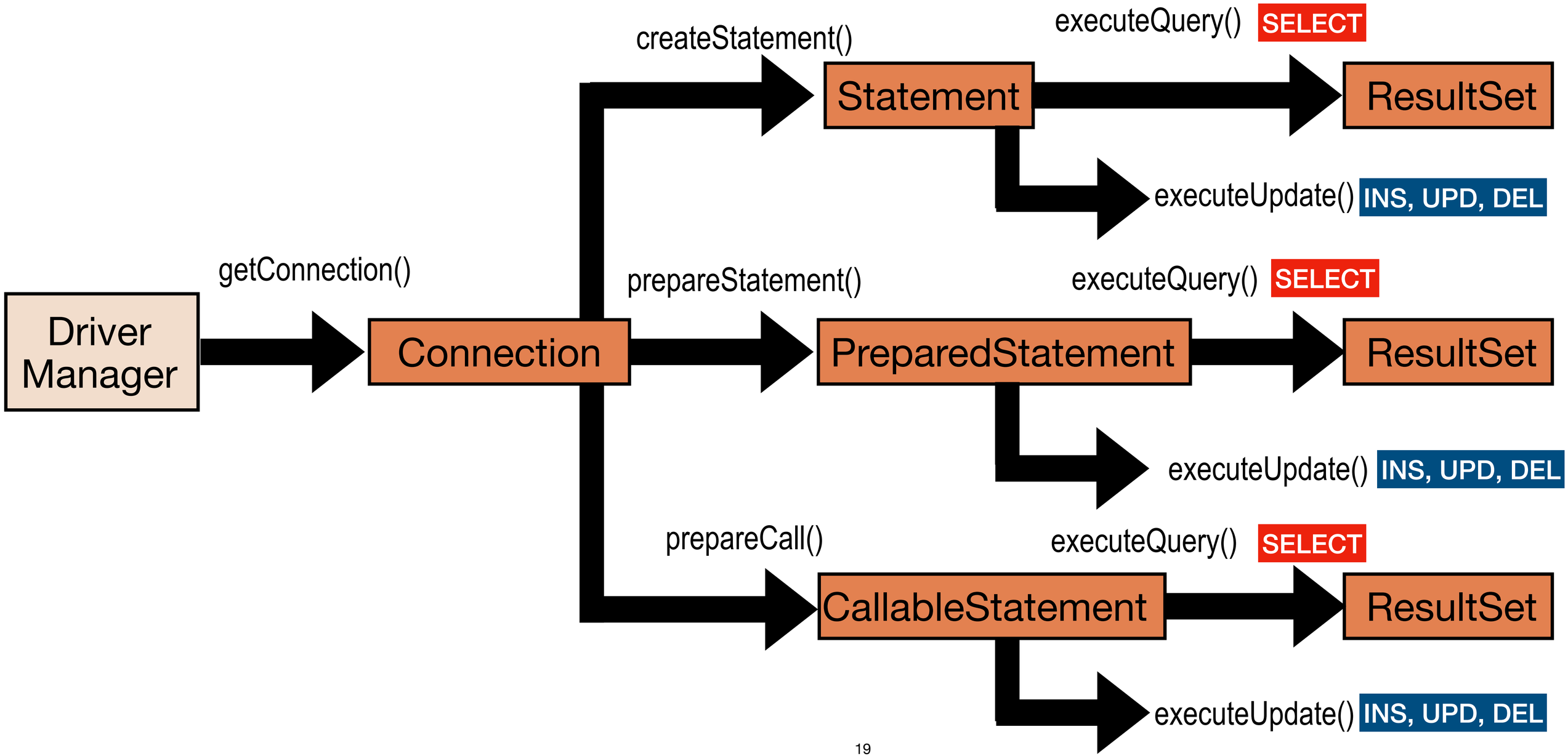
Connection

DriverManager

Class.forName(com.mysql.cj.jdbc.Driver);

Connector/J 9.2.0

Arquitectura JDBC (flujo)



Statement, PreparedStatement y CallableStatement

Statement (Consultas sin parámetros y cuando no se repiten muchas veces)

```
“select nombre from INV_ALMACENES where almacenId = “ + id
```

PreparedStatement (Consultas repetitivas con parámetros dinámicos)

```
“select nombre from INV_ALMACENES where almacenId = ?“
```

CallableStatement (Cuando la lógica está en la base de datos (procedimientos almacenados)

```
“{call SP_REPORTE_ALMACENES(?)}“
```

Statement, PreparedStatement y CallableStatement

- No se recomienda usar la interfaz Statement pue posibilita ataques del tipo **SQL Injection**.
- SQL Injection (inyección SQL) es un **tipo de ataque** que ocurre cuando un atacante inyecta código SQL malicioso en una consulta, con el objetivo de acceder, modificar o eliminar datos de una base de datos.

Statement

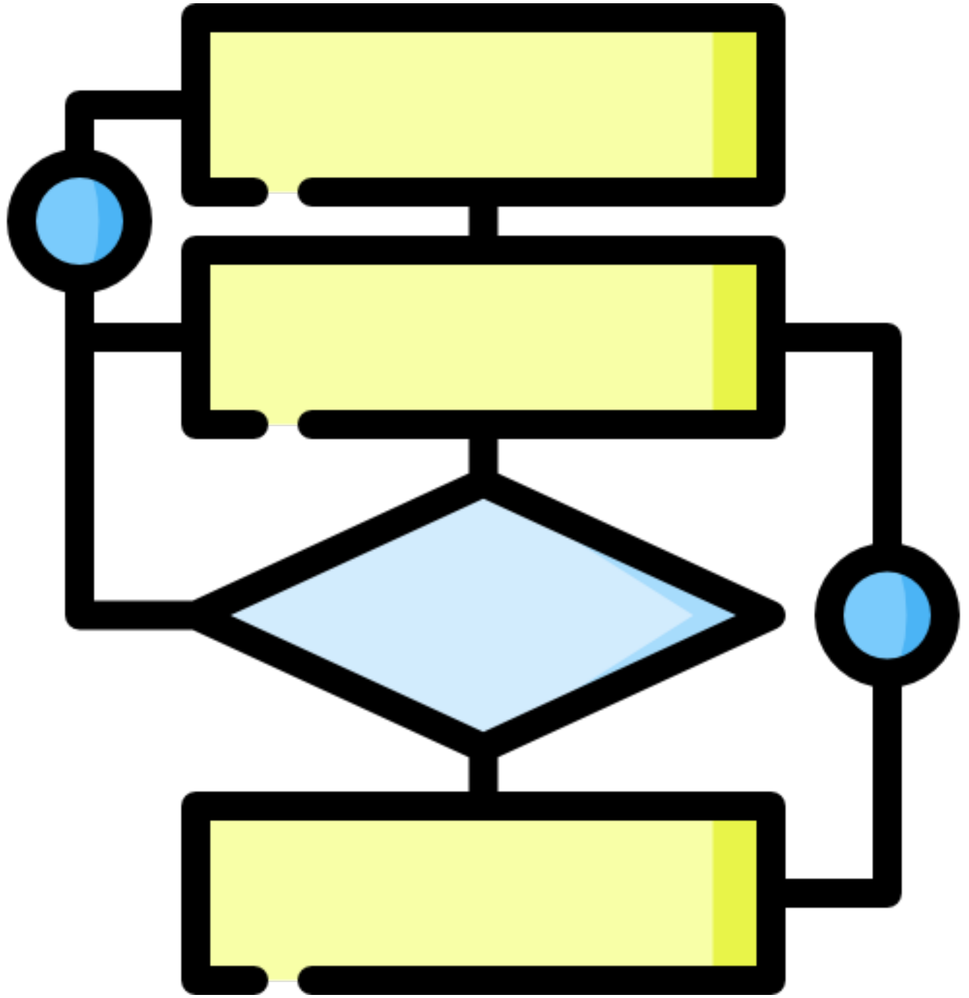
PreparedStatement

CallableStatement

CRUD

Definición

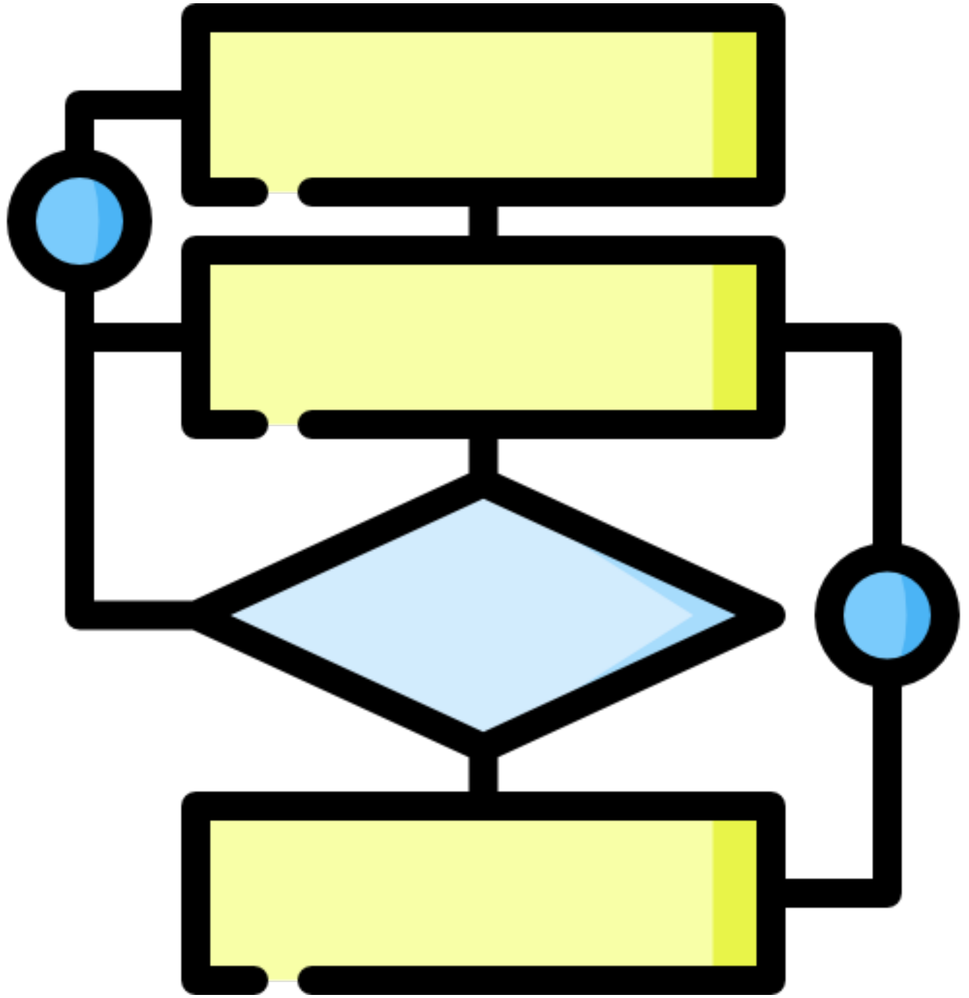
- **CRUD** es un acrónimo que representa las cuatro operaciones básicas que se pueden realizar en una base de datos o en una aplicación que maneja datos:
  - **Create**: Crear un nuevo registro (insert).
  - **Read**: Leer o consultar datos (select).
  - **Update**: Modificar un registro existente (update).
  - **Delete**: Eliminar un registro (delete).



CRUD

Consideración

- Para cada clase del dominio, se gestionará por lo menos las cuatro operaciones básicas, pudiendo incluir más operaciones en caso de ser necesario
  - **Create**:
    - insertar
  - **Read**:
    - obtenerPorId
    - listarTodos
  - **Update**:
    - modificar
  - **Delete**:
    - eliminar



Algoritmo para ejecutar sentencias a través del API JDBC

1. Obtener **conexión** usando para ello el DBManager.

2. **Generar consulta SQL** usando para ello la clase String.

3. Crear **sentencia** basada en la conexión (*Statement*, *PreparedStatement* o *CallableStatement*).

4. **Ejecutar** consulta SQL basada en la sentencia creada (*executeQuery* o *executeUpdate*).

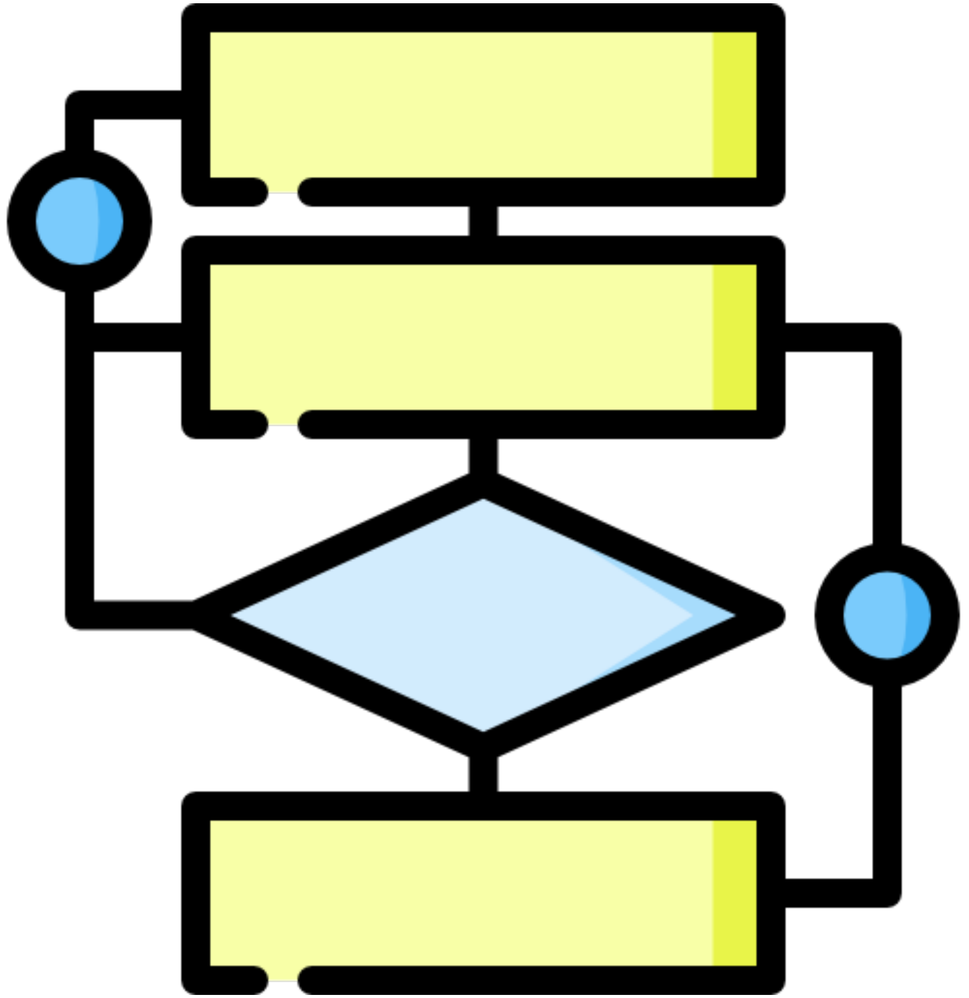
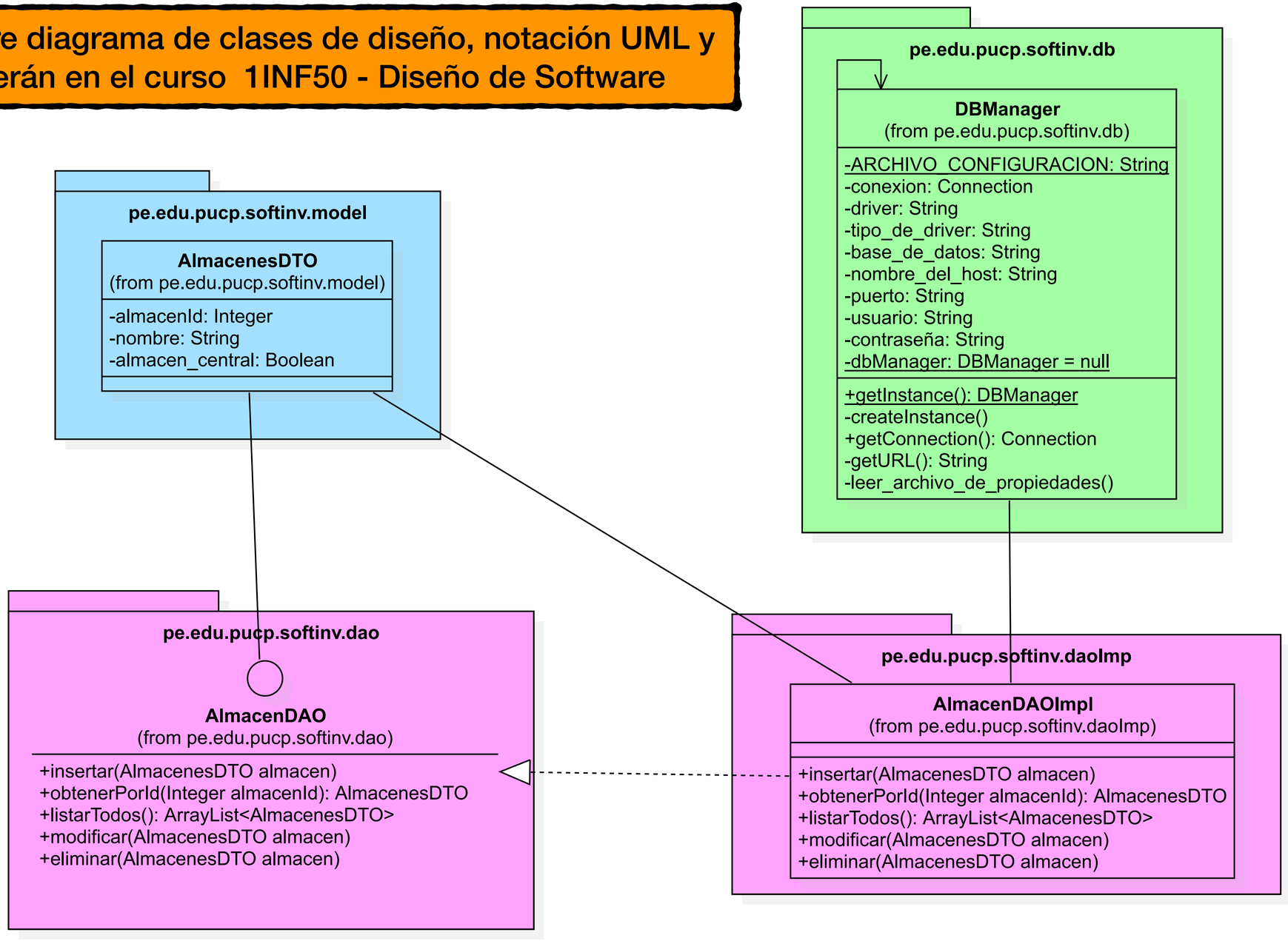
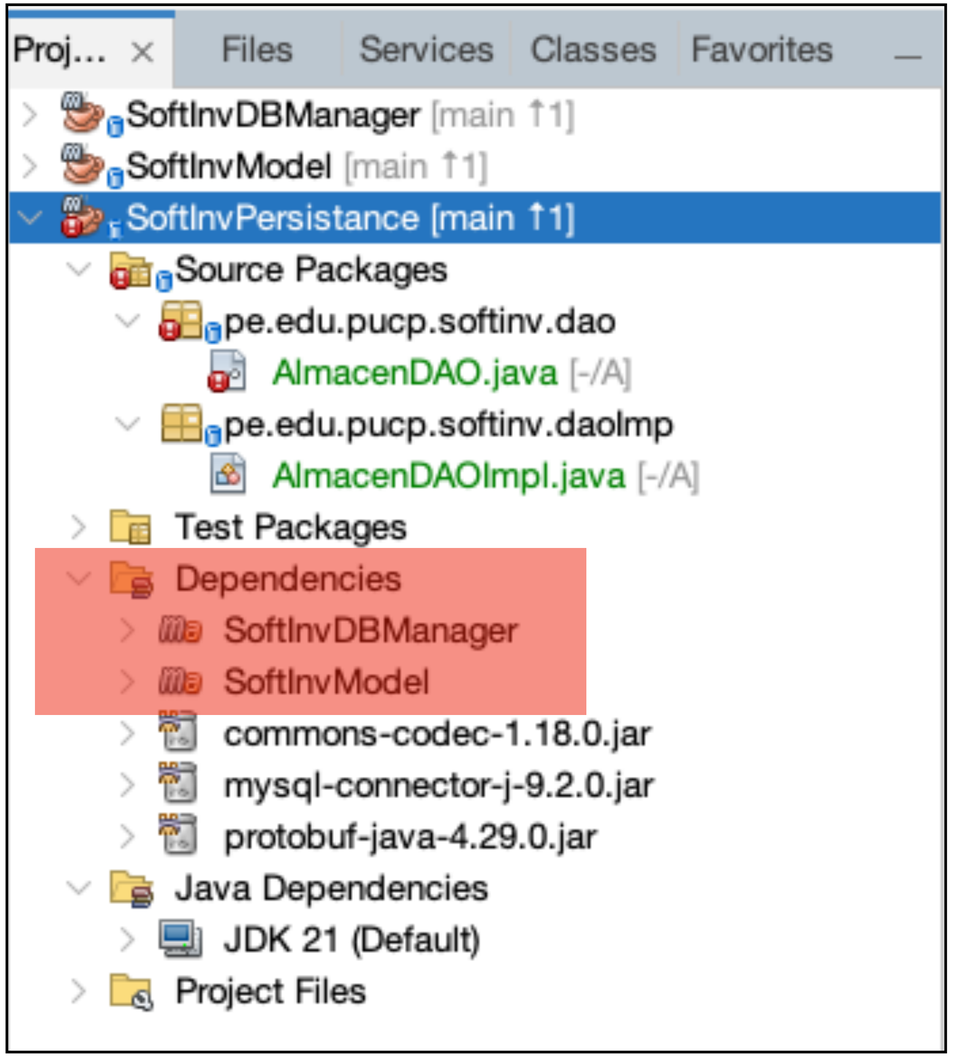


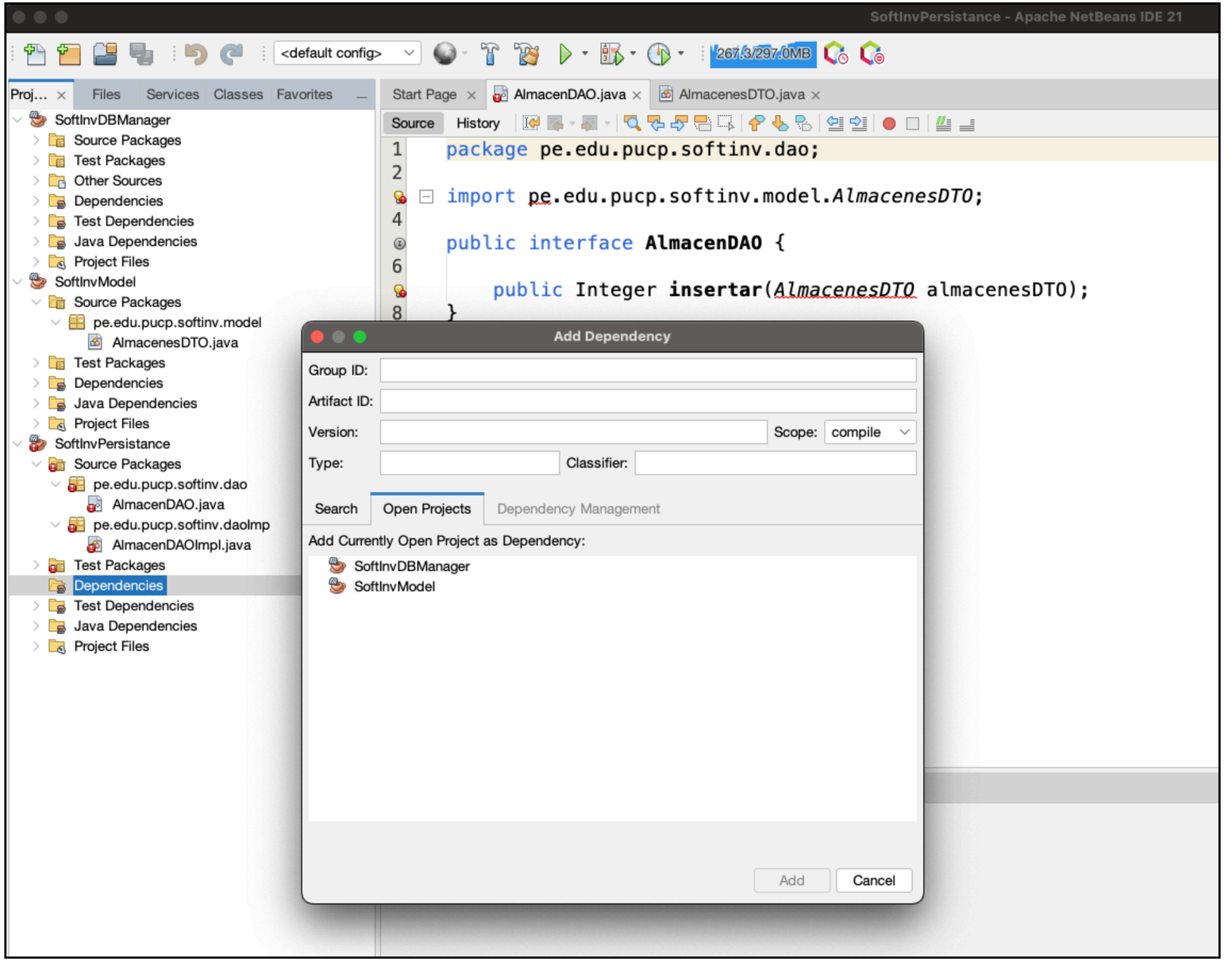
Diagrama de clases (persistencia Almacenes)

Mayor detalle sobre diagrama de clases de diseño, notación UML y patrones, los verán en el curso 1INF50 - Diseño de Software



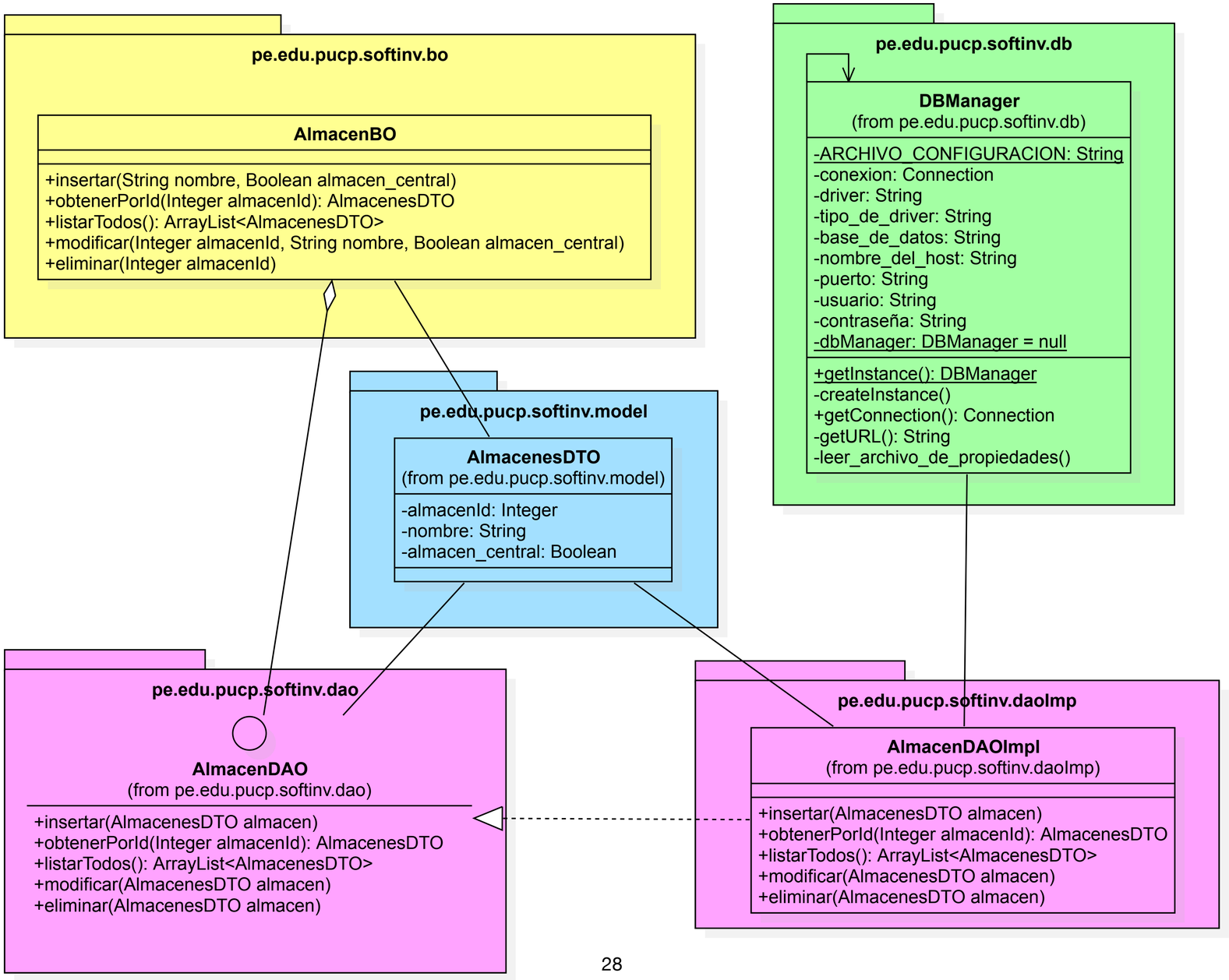
Implementación de la capa de Persistencia





Capa de Negocios

Diagrama de clases (capa de negocio Almacenes)



Tarea

Trabajo para la casa

- Implementar todas las clases de la capa del dominio en el proyecto **SoftInvModel**.
- Para cada entidad del dominio, implementar todas las operaciones CRUD en el proyecto **SoftInvPersistance**, incluyendo las pruebas unitarias (menos ReportesStocksDTO).
- Para cada operación CRUD, implemente su invocación en el proyecto **SoftInvBusiness**, incluyendo las pruebas unitarias.