2025

Programación orientada a objetos con BD

PRODUCTO 4. IMPLEMENTACIÓN MEDIANTE ORM [EVALUACIÓN TEÓRICA]

SERGIO GOMEZ GUTIERREZ

Universitat Oberta de Catalunya

1. **Realiza una comparativa entre las diversas interfaces de programación de persistencia de la aplicación con mapeo ORM, por ejemplo, JPA, Hibernate, JDO. ¿Cuál crees que es el más idóneo para aplicar en este proyecto?**

A. JPA (Java Persistence API).

No es un software que instalas, sino una especificación (un documento). Define cómo se debe guardar la información en Java, pero no hace el trabajo por sí solo.

Es el estándar oficial de Java. Define las anotaciones (como @Entity, @Id) y las interfaces (EntityManager).

B. Hibernate.

Es un framework (una librería de código). Es el "motor" que realmente hace el trabajo de conectarse a la base de datos y generar el SQL de forma automática.

Se considera el ORM más popular y potente en Java. Lo más importante es que Hibernate implementa JPA. Es decir, sigue las normas de JPA, pero añade funciones extra y optimizaciones.

C. JDO (Java Data Objects).

Es otra especificación estándar, diferente a JPA. Su objetivo es guardar objetos java en cualquier sitio (no solo bases de datos SQL, sino también archivos, bases de datos de objetos, etc.

Aunque es muy potente, ha perdido la batalla de la popularidad frente a JPA. Tiene menos documentación y comunidad actualmente.

¿Cuál es el más idóneo para este proyecto?

Hibernate implementando JPA.

Al usar las anotaciones de JPA (el estándar), aprendes la forma oficial de Java. Si en el futuro quieres cambiar Hibernate por otro motor, tu código apenas cambiaría.

Se considera la herramienta más robusta, con más documentación en internet y la más demandada en las ofertas de trabajo actualmente, ya que hibernate gestiona automáticamente las conexiones y transacciones que tanto nos costaron en JDBC.

2. **¿Cómo se implementa ORM?**

La implementación se puede dividir en 3 pasos que sustituyen al código SQL manual:

1- Mapeo (Anotaciones): Se indican anotaciones en las clases Java. Por ejemplo, al poner @Entity sobre la clase Cliente, el ORM sabe automáticamente que se debe crear y gestionar una tabla llamada "Cliente" en la base de datos.

2- Configuración: La conexión a la base de datos (usuario, contraseña, URL) se indica en un archivo de configuración externo, no en el código Java.

3- Gestión: Se utiliza un objeto estándar con el nombre EntityManager. En lugar de escribir sentencias SQL (INSERT, SELECT,etc), le damos órdenes directamente a este objeto usando las clases Java (ej: entityManager.persist(cliente) para guardar).

3. **¿Cómo has integrado el ORM en el MVC?**

La integración del ORM en el patrón MVC se realiza dentro de la capa “Modelo”, actuando como una capa para la abstracción de datos para el resto de la aplicación.

Para implementarla, destaco 2 puntos clave:

1- En las Entidades (Modelo de Datos): Las clases Java (Articulo, Cliente) dejan de ser únicamente objetos y se convierten en entidades persistentes mediante las anotaciones (@Entity, @Id). Ahora estas clases ya saben a qué tabla y columnas de la base de datos corresponden.

2- En los DAOs (Acceso a Datos): Se sustituye el código manual JDBC (SQL, Connection, PreparedStatement) por el EntityManager de JPA.

Antes (JDBC): Escribíamos INSERT INTO...etc. manualmente.

Ahora (ORM): Llamamos a métodos directos como: entityManager.persist(cliente).

Resultado final: El Controlador y la Vista permanecen intactos en el proyecto. El Controlador sigue pidiendo datos al DAO (ej: dao.buscar(id)), sin saber si por detrás hay consultas SQL manuales o si Hibernate está haciendo su trabajo.