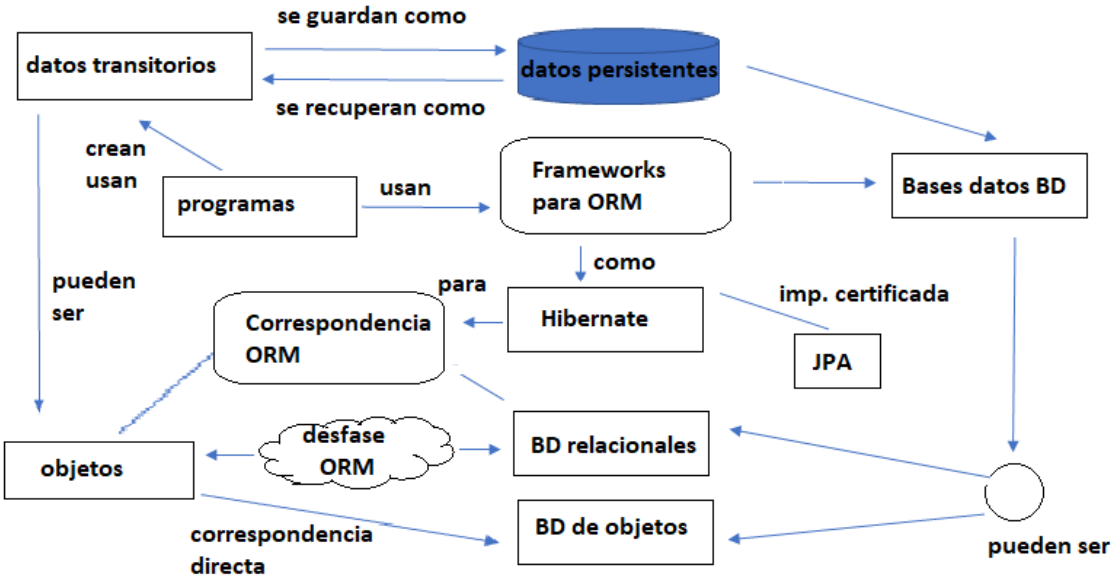
AD3. Acceso a datos con herramientas ORM

Contenido

Introducción:



Resumen de conceptos:

Persistencia de datos: Paso de datos de la memoria principal al medio de almacenamiento secundario (disco) para que puedan volver a ser recuperados a la memoria principal.

Objeto persistente: Objeto que tiene una representación persistente en la base de datos. El objeto persistente está asociado a la detección de los cambios que se realizan sobre él para poder reflejarlos en la base de datos.

Desfase objeto-relacional: Los problemas para la persistencia de objetos en ficheros o en bases de datos relacionales.

ORM (Object Relational Mapping): Correspondencia objeto relacional. Son las técnicas y herramientas que hacen posible la persistencia de objetos en un esquema relacional, teniendo en cuenta la correspondencia con los objetos que representan a las clases en el modelo de programación OO.

JPA (java Persistence Arquitecture): Es la API de java para ORM.

Ejercicio de búsqueda de información:

1- ¿Qué es JDBC?

2- ¿Qué es JPA?

3 - ¿Cuáles son las diferencias entre las conexiones JPA y JDBC?

Comparte el resultado con los compañeros.

https://www.geeksforgeeks.org/difference-between-jdbc-and-hibernate-in-java/

Correspondencia ORM.

El uso de lenguajes OO para trabajar con bases de datos relacionales plantea problemas conocidos como desfase objeto-relacional. La forma de resolver estos problemas se basa en alguno de estos planteamientos:

1. **Bases de datos Objeto Relacionales**:
   Son bases de datos relacionales con capacidad para gestionar objetos. En SQL99 se incluyeron tipos estructurados definidos por el usuario, que pueden ser tipos de objetos (clases). Sin embargo, resuelven parcialmente el problema.
2. **ORM o correspondencia objeto relacional**.
   Se trata de establecer la correspondencia entre las clases definidas en Java y las tablas definidas en la BD relacional. Así como el mecanismo que permite registrar en la base de datos los cambios introducidos en los objetos y recuperar como objetos la información de la base de datos.
   Esta correspondencia hace posible la persistencia de objetos en bases de datos relacionales.
3. **Bases de datos OO**.
   Almacenan objetos directamente. Aunque parece la mejor solución, lo cierto es que este tipo de bases de datos apenas se usa debido a la potencia y fuerza con la que se empezaron a implantar las bases de datos relacionales.

Herramientas ORM para Java:

La primera herramienta ORM para Java posiblemente sea Hibernate, que surgió en el año 2001, como una alternativa a la persistencia de objetos que proporciona Java EE (Java Enterprise Edition) y que está basada en EJB (Enterprise Java Beans).

A partir de ella fueron surgiendo otros frameworsk ORM para Java, por ejemplo:

- Apache Cayenne
- Apache OpenJPA
- Oracle TopLink
- …

El uso de frameworks ORM no está limitado a Java, también se usan en con otros lenguajes como C++, PHP, Python, etc.

Hibernate.

URL: https://hibernate.org/

Fichero hbm o fichero de correspondencia: Son ficheros propios de Hibernate en los que está especificada la correspondencia entre una clase y un esquema relacional (conjunto de tablas asociadas). Es decir, en un fichero de correspondencia se especifica cómo la información contenida en un objeto de una clase determinada se almacena en un esquema relacional.

HQL (Hibernate Query Language): Lenguaje parecido a SQL pero que maneja conjuntos de objetos con sus atributos en lugar de filas y columnas.

JPQL (Java Persistence Query Language): Es un subconjunto de HQL y es parte de la especificación de JPA.

Hibernate es un Framework que surge en el 2001 como una alternativa mejora a Java EE:

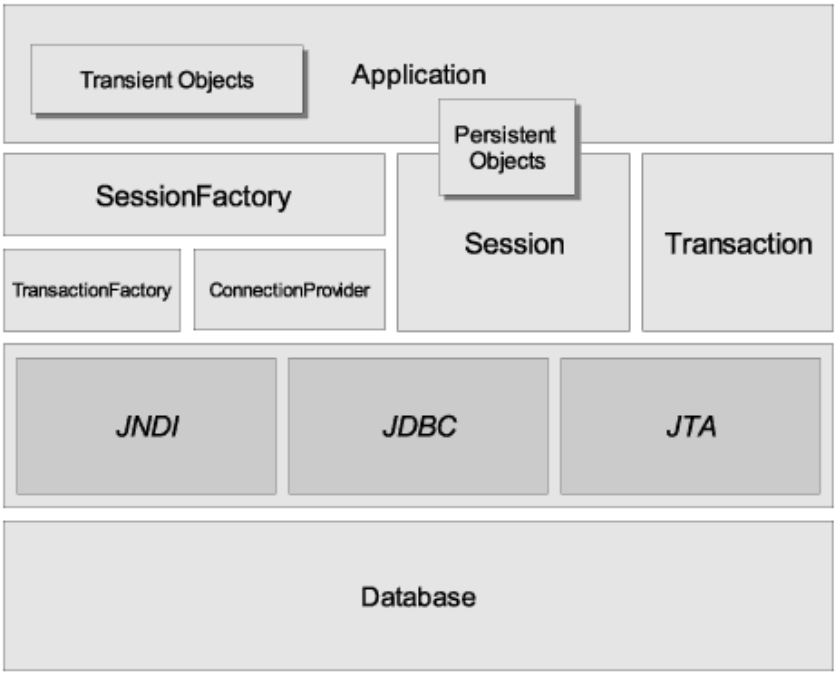
- Se trata de una solución que proporciona persistencia de objetos basada en ORM, pero sin necesidad de un servidor Java EE.
- Soporta transacciones. Utiliza técnicas de alto rendimiento, como el pool de conexiones, técnicas de agrupación por lotes (batching) para las bases de datos.

Hibernate proporciona una API propia como una implementación de la de JPA. Hibernate 3 se convirtió en una versión certificada de JPA en el año 2010. Recordemos que JPA es la API estándar de Java para la persistencia de objetos y parte de la especificación para Java EE.

Hibernate tiene su propio lenguaje de consulta HQL para manejar objetos persistentes:

- Las sentencias SQL actúan sobre filas y columnas de las tablas.
- Las sentencias HQL actúan sobre un conjunto de objetos persistentes y sus atributos.
- JPQL: es el lenguaje estándar de JPA y es un subconjunto de HQL.

Arquitectura de Hibernate:



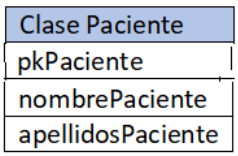
Las aplicaciones que usan Hibernate trabajan con **objetos persistentes** y **objetos transitorios**:

- Los objetos persistentes se asocian a una **session** creada por una **SessionFactory**. La aplicación puede crear objetos transitorios y convertirlos en persistentes para que puedan almacenarse en la BD.
- Las operaciones sobre objetos persistentes se pueden agrupar dentro de transacciones (**Transaction**) que están asociadas a una sesión.
- Las consultas **HQL** se pueden realizar mediante la clase **Query**.

- La interacción con la BD se hacer a través de **la api JPA o JDBC**. Aunque también puede utilizar internamente la API de Java JNDI para fuentes de datos DataSource y JTA para transacciones.

Hibernate: la correspondencia objeto relacional.

Aunque la correspondencia entre objetos Java y datos de una BD relacional es compleja, nosotros empezaremos con el caso más simple, en el que, para cada clase hay una tabla en el modelo relacional que permite almacenar los objetos de esa clase.



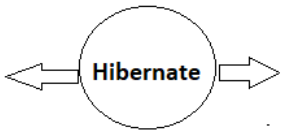


	Tabla paciente	
pkPaciente	nombrePaciente	apellidosPaciente
1	Javier	Abascal
2	Ana	Alonso
3	Oscar	Arregui

A partir de esta correspondencia simple se contemplan otras más complejas que deben representar:

- Colecciones de objetos para las relaciones de 1: N o de N:M en la BD.
- Referencias a objetos para las relaciones 1:1 o de N:1 en la BD.
- Herencia.

IMPORTANTE: Si ya tenemos una base de datos, Hibernate nos permite establecer una correspondencia Objeto Relacional. De esta forma, podemos aprovechar estructuras ya existentes para construir aplicaciones actuales.

Cuando partimos de una Base de Datos conocida, debemos atender a las siguientes condiciones:

- Las tablas deben tener clave primaria.
- Los valores de las columnas que formen parte de la clave primaria no pueden cambiar.
- Las clases deben implementar una interfaz Serializable.
- Las clases deben tener los métodos getX y setX implementados.

Anotaciones en el mapeo.

A la hora de hacer el mapeo, vemos que las anotaciones que aparecen para las clases cuando mapean las tablas y las relaciones utilizan una notación específica:

Las anotaciones que hemos visto cuando en una clase Java que mapeada con una tabla son:

- @Entity Indica que la clase es una tabla en la base de datos
- @Table(name = “nombre_tabla”, catalog = “nombre_base_datos”) Indica el nombre de la tabla y la base de datos a la que pertenece (este último parámetro no es necesario ya que esa información viene en el fichero de configuración)

Y para los atributos simples mapeados con los campos de la tabla correspondiente:

- @Id Indica que un atributo es la clave
- @GeneratedValue(strategy = GenerationType.IDENTITY) Indica que es un valor autonumérico (PRIMARY KEY en MySQL, por ejemplo)

- @Column(name = “nombre_columna”) Se utiliza para indicar el nombre de la columna en la tabla donde debe ser mapeado el atributo.

También se mapean las relaciones, de forma que se puedan implementar o mantener una relación entre clases bidireccional:

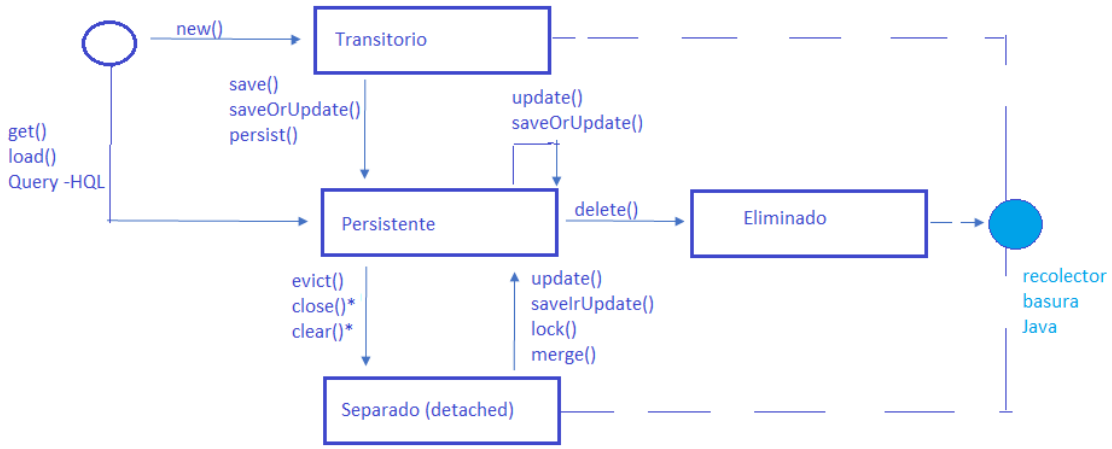
- @OneToOne Indica que el objeto es parte de una relación 1-1
- @ManyToOne Indica que el objeto es parte de una relación N-1. En este caso el atributo sería el lado 1
- @OneToMany Indica que el objeto es parte de una relación 1-N. En este caso el atributo sería el lado N
- @ManyToMany Indica que el objeto es parte de una relación N-M. En este caso se indica la tabla que mantiene la referencia entre las tablas y los campos que hacen el papel de claves ajenas en la base de datos

Sesiones y estados de los objetos persistentes.

Hibernate nos ha permitido realizar la correspondencia ORM para clases de Java y para las relaciones entre ellas. Nos permite crear objetos transitorios que se han almacenado en la BD como objetos persistentes.

La clase para la que establecemos una correspondencia mediante Hibernate se llama clase persistente y podemos crear objetos persistentes instanciando la clase.

Ciclo de vida de los objetos persistentes:



* Afecta a todos los objetos asociados a la sesión

Este ciclo de vida muestra los estados en los que puede estar los objetos persistentes, las operaciones que permiten recuperarlos, modificarlos y grabar de nuevo estas modificaciones en la BD.

Una sesión (interfaz org.hibernate.Session) es esencial en Hibernate porque se construye sobre una conexión a la base de datos.

Las transaccione que hicimos en el ejemplo de clase son transacciones de sesión y pueden incluir varias operaciones sobre la BD. Una sesión puede mejorar el rendimiento de las consultas en objetos persistentes utilizando una caché. La apertura de sesiones se mejora usando un pool de conexiones.

Una sesión, junto con un gestor de entidades asociado, constituye un contexto de persistencia. El gestor de entidades (interfaz javax.persistence.EntityManager) lleva el control de los cambios que se realizan sobre objetos persistentes. La interfaz Session es de la API de Hibernate, mientras que la interfaz EntityManager pertenece a la de JPA. Es posible obtener una EntityManager a partir de una Session y al revés porque hay un puente entre ambas.

Los cambios que hacemos sobre los objetos quedan reflejados en la BD porque están asociados a un contexto de persistencia.

Los métodos save() y saveOrUpdate():

Pasan los objetos de una sesión de transitorios a persistentes.

En clase ya hemos visto como crear un objeto nuevo y hacerlo persistente con save(objeto).

La principal diferencia con saveOrUpdate(objeto) es que permite modificarlo.

Si tratamos de hacer persistente un objeto creado con new() y existe un objeto persistente de la misma clase e idéntico identificador, al hacer save(objeto) generar una excepción porque el identificador es único.

Si utilizamos saveOrUpdate(objeto) para el objeto creado con new(), funcionará como save(objeto). Pero si hay un objeto con el mismo identificador, lo modificará.

Actividad propuesta:

Modifica el nombre de una sede de la BD que estamos usando en la teoría: proyectos.sql

session.saveOrUpdate(sede);

Los métodos get() y load()

Sirven para obtener objetos persistentes.

Ambos métodos se usan para obtener un objeto persistente, recibiendo como argumentos la clase y el identificador único. La diferencia es:

- get(clase, identificador): devolverá null si no existe el objeto con ese identificador
- null(clase, identificador): Eleva ObjectNotFoundException.

El método delete():

Sirve para eliminar un objeto persistente: delete(objeto).

Los métodos close() y clear()

- El método session.clear() → separa todos los objetos asociados a una sesión, sin grabarlos.
- El método session.close() → graba todos los objetos asociados a la sesión y los separa.

Los métodos update(), lock() y merge()

Estos métodos permiten pasar un objeto de separado a persistente.

- Método lock(). Útil cuando hay bloqueos.
- Método update() → Sirve para actualizar un objeto. También podemos usar saveOrUpdate().
- Método merge() → Sirve para actualizar los contenidos de otro objeto persistente, con el mismo identificador en la sesión si el objeto ya existe. En el caso de que no exista un objeto con ese identificador, lo crea.

Ejemplo guiado: Base de datos proyectos.sql

Interfaz QUERY de Hibernate.

La API de Hibernate permite usar el lenguaje HQL para hacer consultas usando la interfaz QUERY que nos resultará útil para hacer consultas y cambios cuando tengamos que localizar objetos persistentes a partir de atributos o campos que no sean identificadores únicos.

La interfaz QUERY también es de JPA. El lenguaje de JPA se conoce como JPQL.

IMPORTANTE: Observa el manual de HQL facilitado en Moodle: https://docs.jboss.org/hibernate/orm/3.5/reference/es-ES/html/queryhql.html