Definición conector:
Conjunto de clases encargadas de implementar una interfaz de programación de aplicaciones (API) y facilitar con ello el acceso a una base de datos para poder conectarse a ella y lanzar consultas.

Podemos encontrarnos con dos normas o protocolos de conexión a BD SQL:

-ODBC(Open Database Connectivity):
Define una API que pueden usar las aplicaciones para abrir una conexión con una BD, enviar consultas, actualizaciones… Fue desarrollado por Microsoft con el objeto de acceder a cualquier dato desde cualquier aplicación.

-JDBC(Java Database Connectivity):
Define una API que pueden usar los programas o aplicaciones Java para conectarse a los servidores de BD relacionales.

Provee:
-interfaz
-arquitectura estándar (permite fabricar drivers/controladores/conectores estandarizados para la conexión de apps Java con BD relacionales)
-acceso a las utilidades de una BD relacional a través de clases e interfaces propias

---

API JDBC es compatible con el modelo de dos capas (app → driver JDBC local → BD) y el modelo de 3 capas (app → capa intermedia [envia comandos SQL y recoge resultados] → driver JDBC remoto → BD)

Desfase Objeto-Relacional:

Paradigma de orientación a objetos VS paradigma del modelo E/R en BBDD →
problemas a la hora de persistir los datos de un programa OO en una BD relacional en filas y tablas.

PROBLEMAS:

1) Tipos de datos a persistir y estructura de datos

IF:
Objetos tienen: estado (valores de los atributos = datos), comportamiento (definido por los métodos que soporta un objeto | las operaciones que puede realizar), identidad (propiedad que lo hace único, debe persistirse o guardarse de forma permanente para poder recuperar la información del objeto a posteriori).

AND:
Una BD relacional solo puede almacenar datos de tipo primitivo en tablas y filas.

THEN:

Debemos convertir los datos de nuestro objeto en datos de tipo primitivo y almacenarlos en las tablas correspondientes – de forma que podamos recuperar esos datos para reconstruír el objeto.

2) Relaciones

OO → establecidas mediante referencias (referencias direccionales) a objetos o colecciones de referencias a objetos. Pueden ser bidireccionales.

E/R → representadas mediante claves foráneas (FK) y copias de valores en distintas tablas. Las FK no son inherentemente direccionales.

3) Navegación

OO → se pasa de un objeto a otro invocando a los métodos que tiran de sus asociaciones.

E/R → para obtener datos de diferentes tablas es necesario hacer al menos una operación Join.

4) Al invocar el lenguaje de consulta SQL desde un lenguaje de programación, se necesita un mecanismo de vinculación que permita recorrer cada una de las filas de una consulta a la BDD.

5) Herencia OO (inherente, normalmente a través de superclases y superclases, con herencia de atributos y métodos de clase padre) != Herencia E/R (no hay herencia de forma nativa, sino estrategias de simulación de estas estructuras) → es necesario usar técnicas para traducir la herencia de un tipo de lenguaje al otro (técnicas utilizadas: 1 tabla por subclase, 1 tabla por jerarquía o 1 tabla por clase).

Concepto de mapeo Objeto/Relacional:
Técnica utilizada en la persistencia de objetos en BD E/R para atenuar el desajuste entre los modelos E/R y OO.

ORM(Object Relational Mapping) = técnica de programación para convertir datos entre un lenguaje de programación OO y un SGBD relacional utilizado en el desarrollo de apps.

Utilidad:
Convierten automáticamente los objetos en los datos primiticos almacenados en las tablas de una BD relacional. Simulan así el acceso a un sistema de BD OO.

Qué sucede en la práctica?
El mapeo O/R crea una BD virtual orientada a objetos sobre la base de datos relacional.

Qué permite del uso de herramientas ORM:

-Uso de características propias de la OO como herencia y polimorfismo.

- Podemos usar las clases de nuestro modelo de dominio, evitando tener que trabajar con filas de tablas de la BD.

- Permite trabajar con distintas BD (SQLServer, Oracle, MySQL, PostGreSQL…), y para cambiar de una a otra BD solo habría que realizar un cambio en el fichero de configuración.

-Genera automáticamente el código SQL necesario para el acceso a la BD, usando el mapeo O/R, que se especifica mediante un documento XML o mediante anotaciones.

-Permite crear, modificar, recuperar y eliminar objetos persistentes y navegar por las asociaciones entre objetos y actualizarlos al final de una transacción.

Ventajas:

-Rapidez en el desarrollo. Por ejemplo, la mayoría de herramientas ORM permiten autogenerar el modelo de dominio a través del script de la BD.
-Abstracción de la base de datos, evitando sentencias SQL embebidas en el código de la aplicación, separándola de esta capa. Esto permite la migración a otro SGBD de forma fácil y sensilla.
-Reutilización.
-Seguridad. Muchas herramientas ORM suelen implementas sistemas para evitar ataques como SQL injections.
-Mantenimiento de código, gracias a la abstracción y ordenación de la capa de datos.
-Lenguaje de consultas propia.

Desventajas:

-Son herramientas complejas que requieren tiempo y esfuerzo para su aprendizaje.
-Aplicaciones más lentas.

Ejemplos de ORM: Hibernate, Doctrine, Propel, SQLAlchemy, Django, SQL Linq, Ibatis...

HIBERNATE

Definición: herramienta ORM para tecnología Java, disponible también para .NET como Nhibernate. Licencia GNU LGPL. Ofrece lenguaje de consulta HQL.

Basado en una filosofía del mapeo O/R partiendo de los POJOS.

Los POJOS son objetos normales de toda la vida di dios de Java. Serializable, con un constructor sin argumentos y con accseso a los campos o propiedades a través de métodos getter y setter. NO SON LO MISMO QUE LOS COMPONENTES JAVA O JAVABEANS.

Los POJOS son almacenados y recupe rados por el programador gracias al motor de Hibernate (objeto especial = sesión).

Arquitectura: dos capas (capa de persistencia y capa de dominio o nejosio/bisnes)

Como fuca la cholla?
Todas las consultas y operaciones de inserción, actualización y supresión (save/update/delete) de datos en la BD se realizan con objetos, en las instancias de las clases mapeadas por Hibernate.

Funcionamiento/archivos/clases:

Motor entre código y base de datos: sesión Hibernate, objeto de tipo Session = servicio o gestor de persistencia que realiza todas las operaciones sobre la BD.

La sesión envuelve una conexión JDBC. Su creación y destrucción es una operación de bajo coste, consume poca memoria. La factoría de objetos Session será un objeto de tipo SessionFactory (idealmente solo puede haber 1). La interfaz Configuration configura el objeto SessionFactory para trabajar con una plataforma de BD determinada. El objeto SessionFactory está disponible para la app para realizar operaciones de persistencia, para lo cual se crea cada vez un objeto Session.

Archivo de configuración c/ extensión cfg.xml: guardan toda la info necesaria para conectarse a la BD (1 archivo de este tipo por BD a la que nos conectemos), además de la ubicación de los archivos de mapeo (hbm.xml) necesarios para establecer la correspondencia entre las clases de la aplicacipn y las tablas de la base de datos.

Consultas hibernate:

1- Método get(): puede retornar un solo objeto ó bien una colección,aplicando el método iterator()

NOTA: get() devuelve null si no encuentra la entidad, load() devuelve
una excepción de Hibernate

2-
a) CreateQuery() retorna una colección con el método list().
b) Puede retornar un solo resultado con método uniqueResult():

3-Método createCriteria(), al igual que el anterior puede retornar una colección ó un solo objeto.

JAVA BEAN

Definición:

Un JavaBean o bean es un componente hecho en software que se puede reutilizar y que puede ser manipulado visualmente por una herramienta de programación en lenguaje Java.
Para ello, se define un interfaz para el momento del diseño (design time) que permite a la herramienta de programación o IDE, interrogar (query) al componente y conocer las propiedades (properties) que define y los tipos de sucesos (events) que puede generar en respuesta a diversas acciones.

Aunque los beans individuales pueden variar ampliamente en funcionalidad desde los más simples a los más complejos, todos ellos comparten las siguientes características:

-Introspection:
Permite analizar a la herramienta de programación o IDE como trabaja el bean

-Customization: El programador puede alterar la apariencia y la conducta del bean.

-Events: Informa al IDE de los sucesos que puede generar en respuesta a las acciones del usuario o del sistema, y también los sucesos que puede manejar.

-Properties: Permite cambiar los valores de las propiedades del bean para personalizarlo (customization).

-Persistence: Se puede guardar el estado de los beans que han sido personalizados por el programador, cambiando los valores de sus propiedades.

En general, un bean es una clase que obedece ciertas reglas:

-Un bean tiene que tener un constructor por defecto (sin argumentos)

-Un bean tiene que tener persistencia, es decir, implementar el interface Serializable.

-Un bean tiene que tener introspección (instrospection). Los IDE reconocen ciertas pautas de diseño, nombres de las funciones miembros o métodos y definiciones de las clases, que permiten a la herramienta de programación mirar dentro del bean y conocer sus propiedades y su conducta.

Propiedades:

Una propiedad es un atributo del JavaBean que afecta a su apariencia o a su conducta. Por ejemplo, un botón puede tener las siguintes propiedades: el tamaño, la posición, el título, el color de fondo, el color del texto, si está o no habilitado, etc.

Las propiedades de un bean pueden examinarse y modificarse mediante métodos o funciones miembro, que acceden a dicha propiedad, y pueden ser de dos tipos:

-getter method: lee el valor de la propiedad

-setter method: cambia el valor de la propiedad.

Propiedades simples: Una propiedad simple representa un único valor.

Propiedades ligadas (bound):

Los objetos de una clase que tiene una propiedad ligada notifican a otros objetos (listeners) interesados, cuando el valor de dicha propiedad cambia, permitiendo a estos objetos realizar alguna acción. Cuando la propiedad cambia, se crea un objeto (event) que contiene información acerca de la propiedad (su nombre, el valor previo y el nuevo valor), y lo pasa a los otros objetos (listeners) interesados en el cambio.

Propiedades restringidas (constrained):

Una propiedad restringida es similar a una propiedad ligada salvo que los objetos (listeners) a los que se les notifica el cambio del valor de la propiedad tienen la opción de vetar (veto) cualquier cambio en el valor de dicha propiedad. (Este tipo de propiedad no se estudiará en este capítulo).