**TEMA 1: JPA**

**¿Qué es JDBC?**

Java Database Connectivity mejor conocida por sus siglas JDBC, es una API que permite enviar comandos SQL desde el lenguaje de programación Java hacia una base de datos relacional, por ejemplo: MySQL, Oracle, Infomix, Sybase etc.

**Funciones del jdbc**

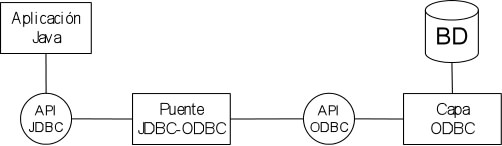
* Establecer la conexión con una base de datos
* Enviar sentencias y/o consultas SQL
* Procesar los resultados obtenidos

**Drivers del jdbc**

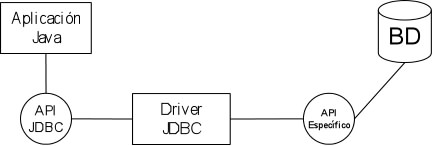
Los drivers del jdbc son adaptadores que el cliente debe tener instalado en su máquina (pc) que permite la traducción de peticiones provenientes desde un programa java para que el protocolo SGBD (sistema gestor de base de datos) pueda reconocerlo y entenderlo.

Los drivers que el jdbc maneja son los siguientes:

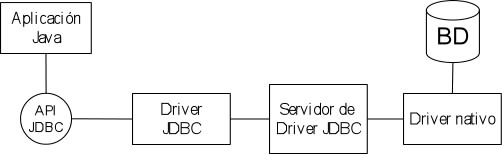
* Driver JDBC Tipo 1 (también llamado Puente JDBC-ODBC) convierte el método JDBC a una llamada a una función ODBC. Utiliza los drivers ODBC para conectar con la base de datos.



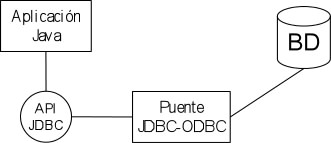
* Driver JDBC Tipo 2 (también llamado driver API-Nativo) convierte el método JDBC a llamadas nativas de la API de la base de datos. Es más rápido que el puente JDBC-ODBC, pero se necesita instalar la librería cliente de la base de datos en la máquina cliente y el driver es dependiente de la plataforma.



* Driver JDBC Tipo 3. Hace uso de un [Middleware](https://es.wikipedia.org/wiki/Middleware) entre el JDBC y el [SGBD](https://es.wikipedia.org/wiki/SGBD).

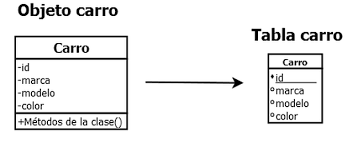


* Driver JDBC Tipo 4 (también llamado Driver Java Puro directo a la base de datos). Es independiente a la plataforma.



**¿Qué es un ORM?**

El mapeo objeto relacional (ORM) es un modelo de programación que consiste en la transformación de las tablas de una base de datos, en una serie de entidades que simplifiquen las tareas básicas de acceso a los datos para el programador, Para esta practica se crean bases de datos orientada a objetos virtual sobre la base de datos relacional esto posibilita que desde java se puedan utilizar las características propias orientada a objetos como la herencia y el polimorfismo.

****

**¿Qué es JPA?**

Java Persistence API conocida por sus siglas como JPA es el standard de Java encargado de automatizar dentro de lo posible la persistencia de nuestros objetos en base de datos.

Conceptos importantes:

**Persistence.xml**

El fichero persistente.xml el cual se encuentra ubicado en el META-INF cumple con dos funciones que son conectarnos a la base de datos y definir las entidades que se van a gestionar.

**EntityManagerFactory**

El EntityManagerFactory es único y es en el que se gestionan las demás entidades en generaly para el uso de varias conexiones se tendrán que crear un nuevo concepto para especificar que ya existen dos entidades del mismo tipo. Este concepto es el que se conoce como PersistenceUnit.

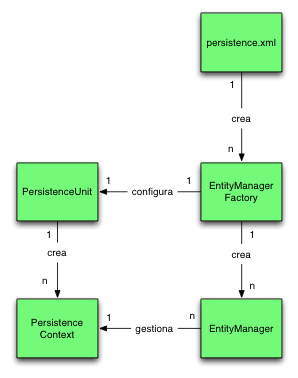
**EntityManager**

Una vez disponemos de un EntityManagerFactory este será capaz de construir un objeto de tipo EntityManager que como su nombre indica gestiona un conjunto de entitades u objetos.

**PersistenceContext**

Para conseguir que alguno de nuestros objetos pase a ubicarse dentro del PersistenceContext bastará con invocar a alguno de los métodos típicos del EntityManager

Para entenderlo mejor podemos apoyarnos en el siguiente diagrama UML:



**¿Qué es hibernate?**

Hibernate es un framework o herramienta de mapeo objeto relacional (ORM) para trabajar en la plataforma de java, esta herramienta ayuda a facilitar el mapeo de atributos entre una base de datos relacional y el modelo de objetos de una aplicación y también agiliza la relación que existe entre la aplicación y la base de datos correspondiente.



**¿Qué es Mapeo Objeto Relacional?**

El mapeo objeto-relacional es una **técnica de programación** que permite convertir datos del sistema de tipos utilizado por los lenguajes de programación orientado a objetos al utilizado por las bases de datos relacionales.

De esta forma, se crea una base de datos **virtual** orientada a objetos sobre la base de datos relacional. Esto, además, posibilita el uso de las características propias de la orientación a objetos (esencialmente la herencia y el polimorfismo).

Las herramientas ORM usan esta técnica para convertir los objetos de la aplicación desarrollada en registros que puedan ser almacenados en una base de datos relacional. Esto suele afectar a un modelo de varias tablas relacionadas por la complejidad que pueda tener el objeto.

De la misma forma, al recuperar por parte de la aplicación los datos almacenados en la base relacional, los registros se convierten en objetos.

De esta forma, **se facilita al programador** la programación abstrayéndolo del modelo relacional de la base de datos

**Ventajas de JPA vs JDBC**

**JPA**

* Nos permite agilizar y facilitar el desarrollo.
* Permite trabajar con la base de datos por medio de entidades en vez de Querys.
* Nos ofrece un paradigma 100% orientado a objetos.
* Elimina errores en tiempo de ejecución.
* Mejora el mantenimiento del software.

**JDBC**

* Ofrece un performance superior ya que es la forma más directa de mandar instrucciones la base de datos.
* Permite explotar al máximo las funcionalidades de la base de datos.
* Posibilidad de utilizar el lenguaje SQL.
* Acceso sencillo a todas las funciones de SQL.
* Los resultados devueltos por SQL pueden ser tratados como objetos, de tal manera que, si hay errores puedan ser tratados como excepciones.

**Principales anotaciones JPA**

Las anotaciones o announces permiten definir la persistencia de forma más fácil entre ellas las más principales son:

**@Entity** Especifica que se va crear una entidad

Ejemplo:

@Entity  
@Table(name=”users”)  
public class User implements Serializable {

@Id  
@GeneratedValue(strategy = GenerationType.IDENTITY)  
@Column(name=”user\_id”, nullable=false)  
private Integer userId = null;

**@Id** Primary key de la entidad

Ejemplo:

@Entity  
@Table (name=” users”)  
public class User implements Serializable {

@Id  
@GeneratedValue (strategy = GenerationType.IDENTITY)  
@Column (name=”user\_id”, nullable=false)  
private Integer userId = null;

**@OneToMany** Indicar relación unidimensional de muchos a uno

Ejemplo:

Ejemplo User N – N Roles

Entity user

@OneToMany (cascade = CascadeType.ALL, mappedBy = “user”)  
private List<UserRole> userRoles = null;

Entity users\_roles

JoinColumn (name = “user\_id”, referencedColumnName = “user\_id”, insertable=false, updatable=false)  
@ManyToOne (optional = false)  
private User user = null;

**@ManyToOne** Podemos indicar desde user\_rols que una relación de esta entidad tiene muchas entidades de roles y usuarios

Ejemplo:

@JoinColumn (name = “user\_id”, referencedColumnName = “user\_id”, insertable=false, updatable=false)  
@ManyToOne (optional = false)  
private User user = null;

@JoinColumn (name = “role\_id”, referencedColumnName = “role\_id”, insertable=false, updatable=false)  
@ManyToOne (optional = false)  
private Role role = null;