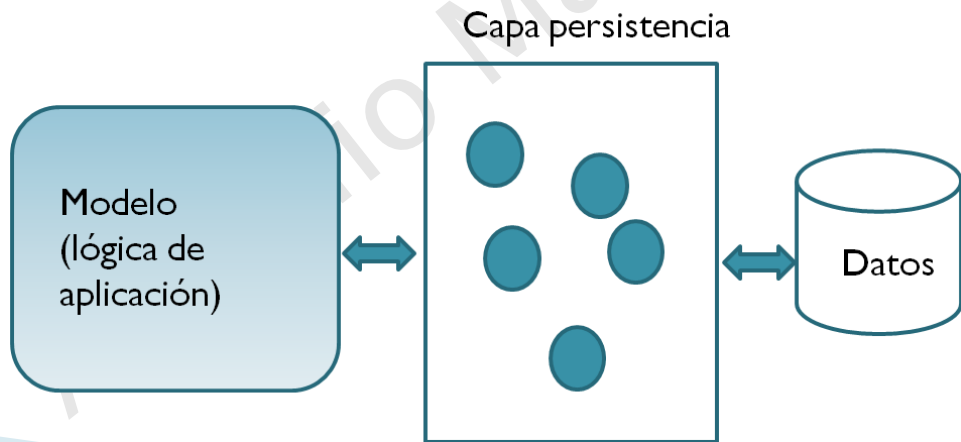


Java Persistence API

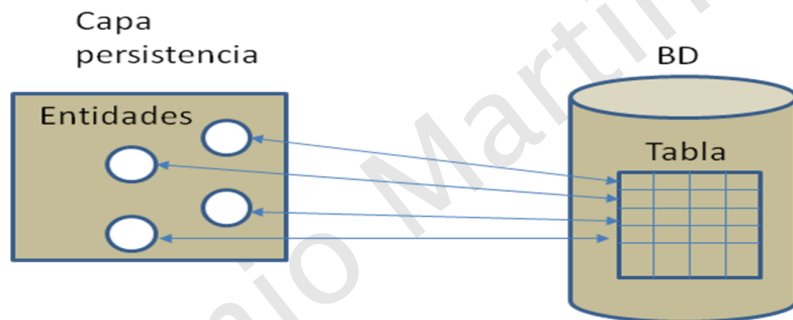
¿Qué es una capa de persistencia?

- Los datos son expuestos a la aplicación en forma de objetos (entidades).
- La lógica de negocio maneja objetos, no tablas



Entidades

- Objetos que representan una fila de una tabla de la base de datos.



- Se definen mediante clases tipo JavaBean
- Las operaciones sobre una capa de persistencia consisten en crear, modificar, eliminar y recuperar entidades.

Frameworks de persistencia

➤ Conjunto de utilidades que facilitan la creación y manipulación de una capa de persistencia.

➤ Entre los más populares:

- Hibernate

- Ibatis

- JPA (no exactamente un framework)

- Componentes de un framework de persistencia

- Motor de persistencia (mapeo ORM)

- Sistema de configuración

Fundamentos JPA

- Especificación Java EE para la creación y gestión de una capa de persistencia
- Clases, interfaces y anotaciones definidas en el paquete `javax.persistence`
- Los principales frameworks de persistencia proporcionan una implementación de esta especificación

Componentes de JPA

- Archivo de configuración persistence.xml:
 - Define unidades de persistencia
 - Cada unidad de persistencia indica el motor utilizado, propiedades de conexión a BD y lista de entidades
- Conjunto de anotaciones para la configuración de entidades (@Entity, @Table, @Id,...)
- API JPA. Conjunto de clases e interfaces para acceso a la capa de persistencia

Creación de una capa de persistencia

- Creación de entidades y configuración a través de anotaciones
- Configuración de la unidad de persistencia a través del archivo persistence.xml o mediante clases de configuración Spring:
 - Proveedor de persistencia
 - Lista de entidades
 - Propiedades de conexión a base de datos

Principales anotaciones JPA

- *@Entity*. Indica que la clase es una entidad
- *@Table*. Mapea la entidad a una tabla de la base de datos
- *@Id*. Indica el atributo que contiene la primary key
- *@Column*. Asocia el atributo con una columna de la tabla. No es necesario si el nombre del atributo coincide con el de la columna.
- *@GeneratedValue*. Indica la estrategia de generación de claves en columnas primary key autogeneradas

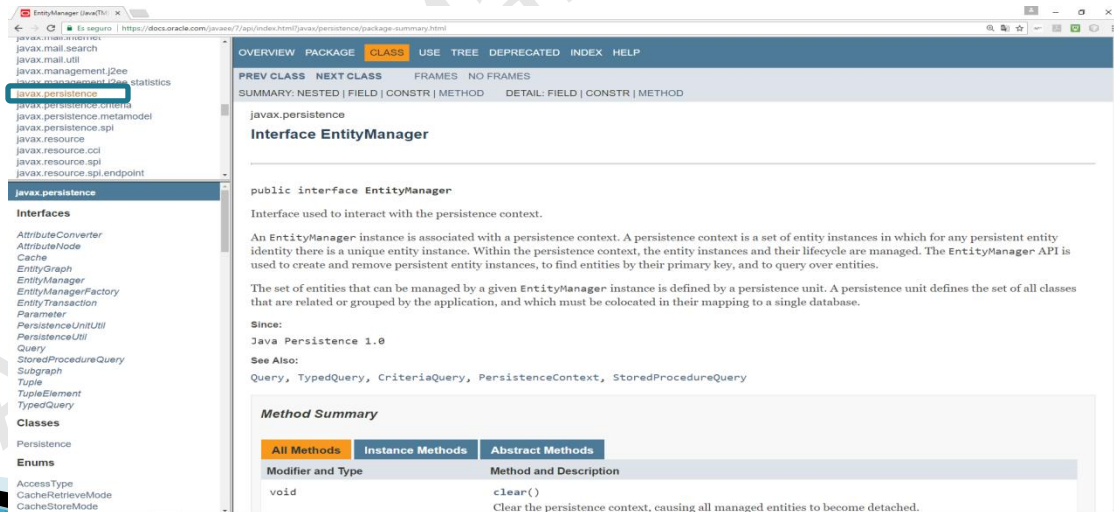
Utilización del API JPA

Javadoc

➤ Ayuda oficial del API:

<https://docs.oracle.com/javase/7/api/index.html?javax/persistence/package-summary.html>

➤ Paquete principal: javax.persistence



The screenshot displays the Oracle Javadoc page for the `javax.persistence` package. The left sidebar shows a tree of packages with `javax.persistence` selected. The main content area shows the `Interface EntityManager` with its description and a `Method Summary` table.

Interface EntityManager

Interface used to interact with the persistence context.

An `EntityManager` instance is associated with a persistence context. A persistence context is a set of entity instances in which for any persistent entity identity there is a unique entity instance. Within the persistence context, the entity instances and their lifecycle are managed. The `EntityManager` API is used to create and remove persistent entity instances, to find entities by their primary key, and to query over entities.

The set of entities that can be managed by a given `EntityManager` instance is defined by a persistence unit. A persistence unit defines the set of all classes that are related or grouped by the application, and which must be colocated in their mapping to a single database.

Since:
Java Persistence 1.0

See Also:
`Query`, `TypedQuery`, `CriteriaQuery`, `PersistenceContext`, `StoredProcedureQuery`

Method Summary

Modifier and Type	Method and Description
void	<code>clear()</code> Clear the persistence context, causing all managed entities to become detached.

Objeto EntityManager

- Implementa la interfaz `javax.persistence.EntityManager`
- Asociado a una unidad de persistencia
- Proporciona métodos para realizar operaciones CRUD sobre la capa de persistencia
- Se obtiene a través de un `EntityManagerFactory` asociado a una unidad de persistencia

Obtención de un EntityManager

➤ Proceso:

- Creación de un objeto EntityManagerFactory a partir de clase Persistence.
- Creación EntityManager desde EntityManagerFactory

```
EntityManagerFactory factory=  
    Persistence.createEntityManagerFactory("unidad_persistencia");  
EntityManager em=factory.createEntityManager();
```

Métodos básicos de EntityManager

➤ Realización de operaciones CRUD:

- `void persist(Object entidad)`. Persiste una entidad, añadiendo sus datos en una nueva fila de la BD.
- `void merge(Object entidad)`. Actualiza la BD con los datos de la entidad, que debe formar parte de la unidad de persistencia
- `T find(Class<T> clase_entidad, Object key)`. Recupera una entidad a partir de su clave primaria
- `void remove (Object entidad)`. Elimina la entidad

➤ `void refresh(Object entidad)`. Actualiza las propiedades de la entidad con los valores de la BD

Transacciones

- Las operaciones de acción se deben englobar en una transacción
- Gestionadas mediante `EntityTransaction` , que se obtiene desde `EntityManager`

```
EntityTransaction tx=em.getTransaction();
```

- Métodos para gestionar la Transacción:
 - `begin()`. Inicia transacción
 - `rollback()`. Rechaza transacción
 - `commit()`. Confirma transacción

Integración JPA-Hibernate con Spring

Características

- La configuración de persistencia se incluye en la configuración Spring. No es necesario persistence.xml
- Inyección de dependencia de EntityManager
- Gestión automática de la transaccionalidad
- Se debe incorporar el módulo Spring ORM

Capa de repositorio

- Incluirá las instrucciones JPA de manipulación de entidades
- Mediante la anotación `@PersistenceContext` se inyectará el `EntityManager`
- Mediante la anotación `@Transactional` se aplicará la transaccionalidad en los métodos de acción

Configuración

➤ En el archivo de configuración del modelo, será necesario definir los siguientes objetos:

- `DataSource`. Puede utilizarse el integrado en Spring o el del servidor de aplicaciones
- `HibernateJpaVendorAdapter`. Adaptador Spring para integrar Hibernate
- `LocalContainerEntityManagerFactoryBean`. Objeto que configura la información de persistencia con JPA. Requiere los dos anteriores
- `JpaTransactionManager`. Gestor de transacción de la capa de persistencia

Consultas JPA

Fundamentos

- Permiten recuperar, actualizar y eliminar entidades en base a diferentes criterios
- Se definen mediante un lenguaje especial llamado JPQL, similar a SQL
- Las consultas son gestionadas a través de un objeto de la interfaz Query o TypedQuery, obtenido a partir de EntityManager

El lenguaje JPQL

➤ Adaptación de SQL para tratar con entidades

➤ Formato instrucciones:

- `select alias from Entidad alias where condicion`

- `delete from Entidad alias where condicion`

- `update Entidad alias set alias.atributo=valor`

➤ Ejemplos:

```
select c from Contacto c
```

```
select e from Empleado e where e.dni='5555K'
```

```
update Empleado e set e.salario=e.salario*1.05
```

```
delete from Curso c where c.nombre like 'P%'
```

Objeto Query

- Implementación de la interfaz `javax.persistence.Query` que permite lanzar consultas JPQL a la capa de persistencia
- Se obtiene a partir del método `createQuery()` de `EntityManager`:

```
String jpql="select c from Contacto c";  
Query qr=em.createQuery(jpql);
```

Métodos de Query

- **List getResultList().** Devuelve una colección List con las entidades recuperadas por la consulta:

```
String jpql="select c from Contacto c";  
Query qr=em.createQuery(jpql);  
//casting al tipo de colección específica  
List<Contacto> contacts=(List<Contacto>)qr.getResultList();
```

- **Object getSingleResult().** Devuelve la única entidad resultante de la consulta. Si la consulta devolviera más de un resultado, se producirá una excepción
- **int executeUpdate().** Ejecuta la consulta cuando se trata de una instrucción de acción (Delete o Update)

TypedQuery

➤ Subinterfaz de Query que permite especificar el tipo de entidad de la respuesta:

▪ **List<T> getResultList().** Devuelve una colección del tipo de las entidades:

```
String jpql="select c from Contacto c";  
TypedQuery<Contacto> qr=em.createQuery(jpql,Contacto.class);  
//No hay que hacer casting  
List<Contacto> contacts=qr.getResultList();
```

▪ **T getSingleResult().** Devuelve la única entidad resultante de la consulta. Si no hay resultados o hay más de uno, genera una excepción

➤ Disponible desde JPA 2

Parámetros en JPQL

➤ Una consulta JPQL puede incluir parámetros en aquellos valores que son desconocidos.

➤ Se pueden definir parámetros por posición o nombre:

```
String jpql="select c from Contacto c where c.edad=?1";
```

```
String jpql2="select e from Empleado where e.departamento=:dep";
```

➤ Para sustituir los valores de los parámetros se utiliza el método `setParameter` de Query:

```
TypedQuery<Contacto> query=em.createQuery(jpql,Contacto.class);  
query.setParameter(1,35);  
List<Contacto> contactos=query.getResultList();
```

```
TypedQuery<Empleado> query=em.createQuery(jpql2,Empleado.class);  
query.setParameter("dep","RRHH");  
List<Empleado> empleados=query.getResultList();
```

Consultas de acción

- Se ejecutan con el método `executeUpdate()` de `Query`, que devuelve el total de entidades afectadas.
- Deben ser englobadas dentro de una transacción:

```
EntityTransaction tx=em.getTransaction();  
String jpql="Delete from Contacto c where c.email like=?1";  
Query query=em.createQuery(jpql);  
query.setParameter(1,"%com");  
tx.begin();  
int res=query.executeUpdate();  
tx.commit();
```

NamedQueries

➤ Una NamedQuery es una instrucción JPQL que se define en la entidad en lugar de en el código, lo que permite reutilizarla en varios puntos de la aplicación.

Se pueden definir varias consultas, cada una con su correspondiente anotación

Definición

```
@Entity
@Table(name="contactos")
@NamedQuery(name="Contacto.findAll",
            query="select c from Contacto c")
public class Contacto{

}
```

Uso

```
TypedQuery<Contacto> query;
query=em.createNamedQuery("Contacto.findAll",Contacto.class);
:
```