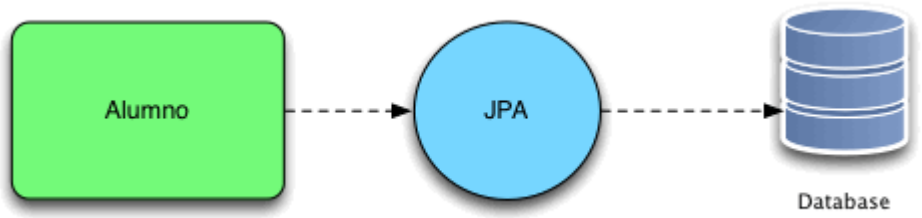
Poco a poco JPA se va afianzando como standard y cada día es más importante conocerlo en mayor profundidad. Una de las características que pueden sernos interesantes en un momento de terminado son los Entity Listener. Estas clases nos permiten escuchar y reaccionar a las distintas operaciones que realiza un EntityManager sobre una entidad en concreto. Vamos a ver un ejemplo sencillo partiendo de la Entidad Alumno:

```
package com.arquitecturajava;

import java.io.Serializable;

import javax.persistence.Entity;
import javax.persistence.EntityListeners;
import javax.persistence.Id;

@Entity
public class Alumno implements Serializable{

private static final long serialVersionUID = 1L;
@Id
private String dni;
private String nombre;
private String apellidos;
private int edad;

public Alumno() {
super();
}
public Alumno(String dni, String nombre, String apellidos, int edad) {
super();
```

```
this.dni = dni;
this.nombre = nombre;
this.apellidos = apellidos;
this.edad = edad;
}

public String getDni() {
return dni;
}
public void setDni(String dni) {
this.dni = dni;
}
public String getNombre() {
return nombre;
}
public void setNombre(String nombre) {
this.nombre = nombre;
}
public String getApellidos() {
return apellidos;
}
public void setApellidos(String apellidos) {
this.apellidos = apellidos;
}
public int getEdad() {
return edad;
}
public void setEdad(int edad) {
this.edad = edad;
}
}
```


Hemos creado la clase y ahora vamos a utilizar JPA para almacenar un nuevo registro en la base de datos:



Para ello nos apoyaremos en un EntityManagerFactory y un EntityManager:

```
package com.arquitecturajava.main;

import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.EntityTransaction;
import javax.persistence.Persistence;

import com.arquitecturajava.Alumno;

public class Principal {

public static void main(String[] args) {
```

```
EntityManagerFactory emf =
Persistence.createEntityManagerFactory("UnidadCurso");
EntityManager em = emf.createEntityManager();
EntityTransaction transaccion= em.getTransaction();
transaccion.begin();
Alumno pedro= new Alumno("1","pedro","gomez",30);
em.persist(pedro);
transaccion.commit();
em.close();

}

}
```

Acabamos de insertar un nuevo registro:

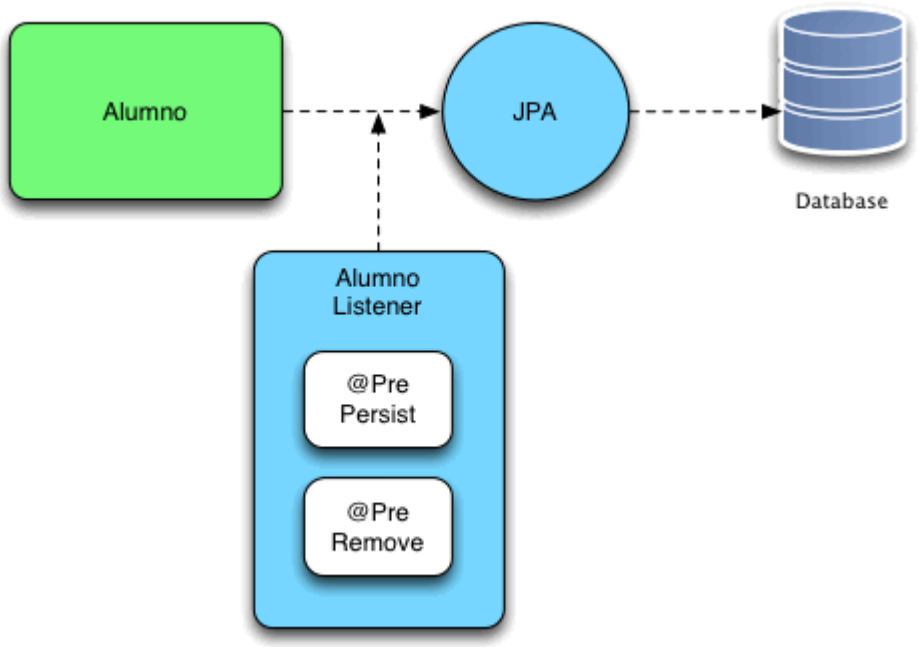
```
INFO: HHH000046: Connection properties: {user=root, password=****, autocommit=true, release_mode=auto}
ago 18, 2014 1:28:18 PM org.hibernate.dialect.Dialect <init>
INFO: HHH000400: Using dialect: org.hibernate.dialect.MySQLDialect
ago 18, 2014 1:28:18 PM org.hibernate.engine.transaction.internal.TransactionFactoryInitiator initiateService
INFO: HHH000268: Transaction strategy: org.hibernate.engine.transaction.internal.jdbc.JdbcTransactionFactory
ago 18, 2014 1:28:18 PM org.hibernate.hql.internal.ast.ASTQueryTranslatorFactory <init>
INFO: HHH000397: Using ASTQueryTranslatorFactory
Hibernate: insert into Alumno (apellidos, edad, nombre, dni) values (?, ?, ?, ?)
```

Entity Listener

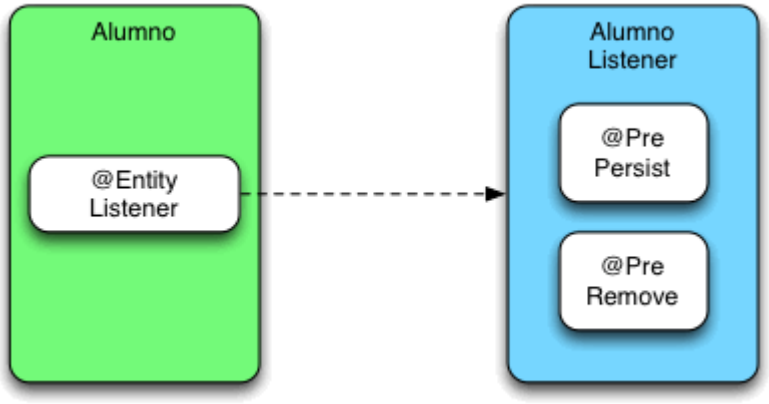
Es en este tipo de casuística donde nos podemos encontrar con la necesidad de realizar alguna operación antes de insertar,actualizar o borrar. Vamos a ver como usar un EntityListener. Para ello nos creamos una nueva clase que la denominaremos AlumnoListener:

```
package com.arquitecturajava;

import javax.persistence.PrePersist;
import javax.persistence.PreRemove;

public class AlumnoListener {

@PrePersist
public void salvar(Alumno a) {

System.out.println("nuevo alumno :" + a.getNombre() + a.getApellidos()
+ a.getEdad());

}

@PreRemove
public void borrar(Alumno a) {

System.out.println("borramos alumo" + a.getNombre() + a.getApellidos()
+ a.getEdad());

}

}
```

Acabamos de implementar dos de los callback de los EntityListener. El que corresponde a @PrePersist y el que corresponde a @PreRemove. Que como su nombre indica son los correspondientes a añadir un listener antes de insertar la Entidad en la base de datos y antes de borrar (existen varios tipos más). Una vez definido el listener y sus métodos de callback este se ubicará en medio del proceso de persistencia.



Para que esto funcione correctamente deberemos registrar el EntityListener para esa Entidad en concreto.



Vamos a ver como queda modificado el código fuente de la clase Alumno:

```
@Entity
@EntityListeners({AlumnoListener.class})
public class Alumno implements Serializable{

// codigo
}
```

Hemos añadido la anotación @EntityListeners que se encarga de registrar el Listener para la clase Alumno cada vez que ahora insertemos un nuevo Alumno el Listener se ejecutará:

```
INFO: HHH000397: Using ASTQueryTranslatorFactory
nuevo alumno :pedrogomez30
Hibernate: insert into Alumno (apellidos, edad, nombre, dni) values (?, ?, ?, ?)
```

Los EntityListener nos pueden ayudar en muchas ocasiones aportando flexibilidad

Otros artículos relacionados: Flexibilidad en JPA ,EntityManager y Singletons ,EntityManagerEstados