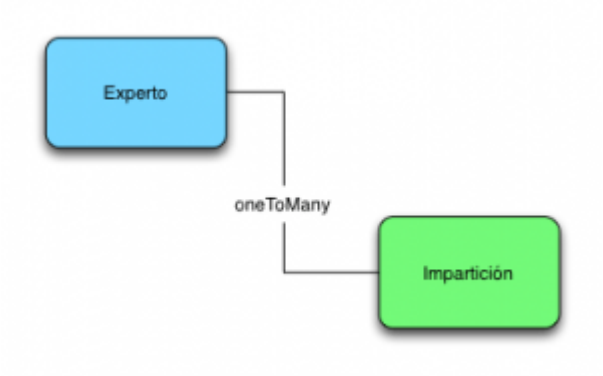
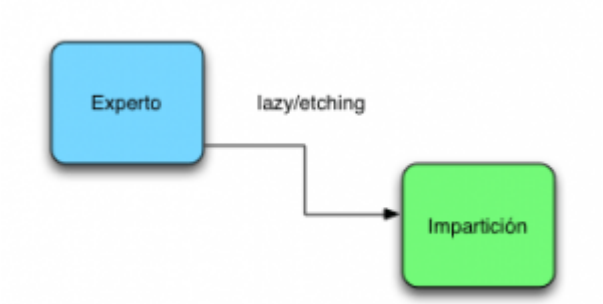
El concepto de JPA Proxy es a veces difícil de entender para la gente que comienza a trabajar con JPA. ¿Cómo funciona un JPA Proxy? .Vamos a apoyarnos en el ejemplo del artículo anterior y hacer una pequeña modificación . Recordemos que partimos de dos clases Experto e Imparticion relacionadas a través de una relación oneToMany.

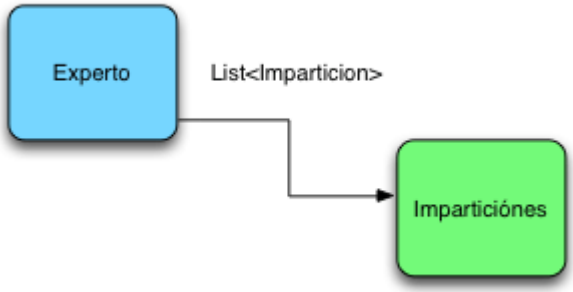


Este tipo de relaciones están definidas a través de JPA como relaciones Lazy Feching. Es decir solo se obtienen las Imparticiones cuando las solicitamos explicitamente.

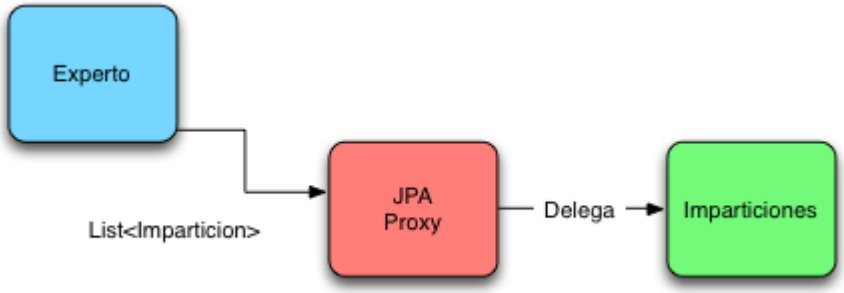


JPA Proxy

Ahora bien la relación es algo más compleja de lo que parece en un primer momento ya que esta ligada a un JPA Proxy. Vamos a intentar aclararlo más, si nos fijamos de una forma más cercana en la relación nos podremos dar cuenta que realmente el concepto de Experto esta relacionado con una lista de Imparticiones List<Imparticion>.



Es aquí donde hay que darse cuenta de que la relación no existe tal como la dibujamos sino que JPA añade un intermediario, un proxy que se encarga de cargar los datos cuando realmente los solicitemos.



Es relativamente sencillo ver que las cosas funcionan así simplemente podemos reutilizar el programa que hicimos en el artículo anterior e invocar al método getClass().getName() del API de Reflection para que nos imprima por pantalla cual es la clase real que la lista de imparticiones tiene.

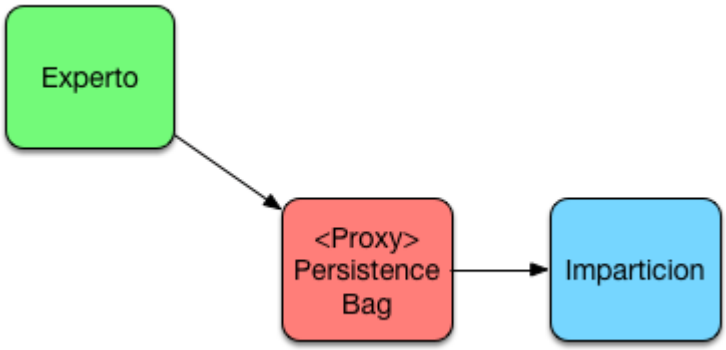
```
package com.arquitecturajava;

import java.util.List;
```

```
import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.Persistence;
import javax.persistence.TypedQuery;

public class Principal {

	public static void main(String[] args) {

		EntityManagerFactory emf =
Persistence.createEntityManagerFactory("UnidadCharla");
		EntityManager em = emf.createEntityManager();
		TypedQuery<Experto> consulta = em.createQuery("select
e from Experto e", Experto.class);

		List<Experto> lista = consulta.getResultList();

		for (Experto e : lista) {

			System.out.println(e.getNombre());
System.out.println(e.getImparticiones().getClass().getName());
			for (Imparticion i : e.getImparticiones()) {
				System.out.println(i.getTitulo());
			}

		}
		em.close();

	}

}
```

Si vemos el resultado por pantalla nos sorprenderá:

```
Hibernate: select experto0_.nombre as nombre1_0_ from Experto experto0_
pedro
org.hibernate.collection.internal.PersistentBag
Hibernate: select imparticio0_.nombre_experto as nombre_e4_1_0_, imparticio0_.id
 as id1_1_0_, imparticio0_.id as id1_1_1_, imparticio0_.nombre_experto as nombre
_e4_1_1_, imparticio0_.fecha as fecha2_1_1_, imparticio0_.titulo as titulo3_1_1_
 from Imparticion imparticio0_ where imparticio0_.nombre_experto=?
```

La clase que almacena la lista de Imparticiones no es un ArrayList como cabría esperar sino que es un PersistenceBag una clase que Hibernate aporta como JPA Proxy.



En el día a día ni siquiera nos daremos cuenta de que la usamos , pero está y realiza su función.

Otros artículos relacionados: Introducción a JPA , JPA Embedded Object , Spring Data y JPA