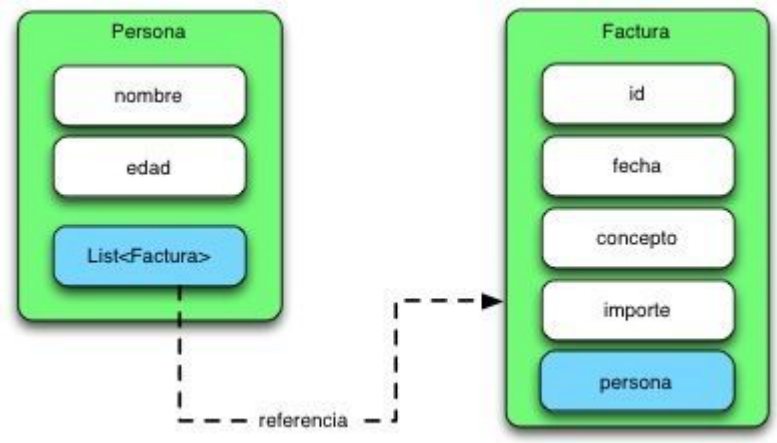
JPA @OneToMany es una anotación clave . En el post anterior hemos trabajado con el concepto de Persona y Factura construyendo una relación entre la Factura y la Persona a la que la factura pertenece con @ManyToOne . En este post construiremos la relación del otro lado entre la Persona y la Factura usando JPA @OneToMany. Para ello lo primero que tendremos que tener es una relación entre ambas clases.



Para ello el primer paso será añadir una variable de tipo List<Factura> dentro de la clase Persona.

```
public class Persona {

	private String nombre;
	private int edad;

	List<Factura> milista= new ArrayList<Factura>();
```

```
public Persona() {
        super();
        // TODO Auto-generated constructor stub
}
public Persona(String nombre, int edad) {
        super();
        this.nombre = nombre;
        this.edad = edad;
}
public String getNombre() {
        return nombre;
}
public void setNombre(String nombre) {
        this.nombre = nombre;
}

public int getEdad() {
        return edad;
}
public void setEdad(int edad) {
        this.edad = edad;
}
```

El siguiente paso será anotar la clase de la forma correcta para que soporte la relación. En este caso utilizaremos la anotación JPA @OneToMany.

Vamos a verlo en código :

```
@Entity
public class Persona {

	@Id

	private String nombre;

	private int edad;

	@OneToMany(mappedBy="persona" )
	private List<Factura> listaFacturas;

	public Persona() {
		super();
		listaFacturas=new ArrayList<Factura>();
	}

	public List<Factura> getListaFacturas() {
		return listaFacturas;
	}
```

```
public void addFactura(Factura f) {

        listaFacturas.add(f);
}

public void setListaFacturas(List<Factura> listaFacturas) {
        this.listaFacturas = listaFacturas;
}

public Persona(String nombre, int edad) {
        super();
        this.nombre = nombre;
        this.edad = edad;
        listaFacturas=new ArrayList();
}
public String getNombre() {
        return nombre;
}
public void setNombre(String nombre) {
        this.nombre = nombre;
}

public int getEdad() {
        return edad;
}
public void setEdad(int edad) {
        this.edad = edad;
}
```

JPA @OneToMany y MappedBy

Todo parece bastante sencillo . Ahora bien tenemos un problema con la anotación JPA @OneToMany y es que tiene un parámetro que se denomina mappedBy y cuyo valor es "persona". Este parámetro hace referencia a que la relación complementaria de ManyToOne ya fue construida por la otra clase "Factura" a través de su variable "persona" como podemos ver a continuación:

```
package es.curso.bo;

import java.util.Date;

import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.JoinColumn;
import javax.persistence.ManyToOne;

@Entity
public class Factura {

    @Id
    private int id;
    private Date fecha;
    private double importe;
    private String concepto;

    @ManyToOne
    @JoinColumn(name="persona_nombre")
    private Persona persona;

    public Factura() {
```

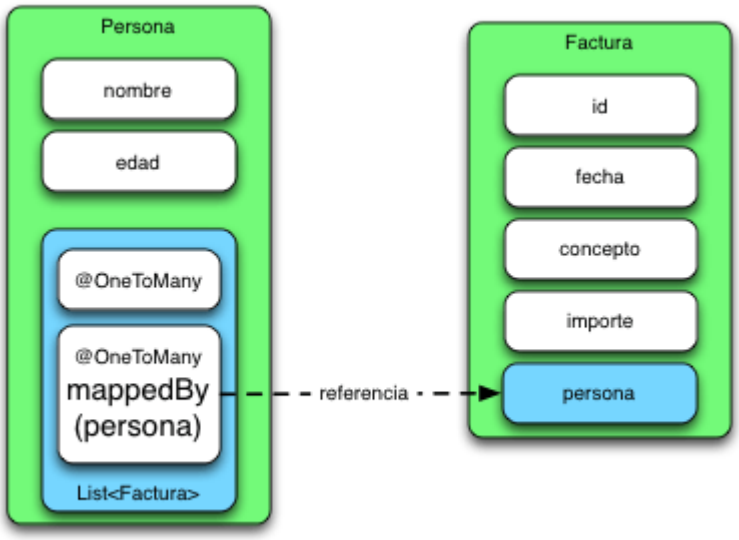
```
        super();
        // TODO Auto-generated constructor stub
    }

    public Factura(int id, Date fecha, double importe, String
concepto,Persona persona) {
        super();
        this.id=id;
        this.fecha = fecha;
        this.importe = importe;
        this.concepto = concepto;
        this.persona=persona;
    }

    public Persona getPersona() {
        return persona;
    }

    public void setPersona(Persona persona) {
        this.persona = persona;
    }
```

Esto normalmente genera dudas a la hora de crear las relaciones. La siguiente imagen intenta clarificar el concepto. El atributo mappedBy hace referencia a la relación entre “clases”.



Por otro lado es el atributo @JoinColumn el que define que columna en la tabla Facturas hace efectiva la relación entre ambas clases de tal forma que toda la información quede correctamente configurada.

Conclusiones

Entendido este concepto hemos terminado de construir una relación bidireccional entre ambas clases y las podemos usar de forma natural con JPA.

Otros artículos relacionados

1. Un ejemplo de JPA Entity Graph
2. JPA (III) EntityManager métodos

3. JPA @ ManyToOne
4. JPA vs Spring Data y sus diferencias
5. Spring Boot JPA y su configuración
6. JPA Query Language Objetos vs Tablas
7. ¿JPA vs Hibernate?

Cursos gratuitos relacionados

1. Introducción a JPA