Tabla de contenido

One2one	CLASE One	CLASE one
UNIDIRECCIONAL	@Entity public class Persona implements Serializable { @Id private long id; @OneToOne(cascade={CascadeType.PERSIST, CascadeType.REMOVE}) private Direccion direccion;	@Entity public class Direccion implements Serializable { @Id @GeneratedValue(strategy=GenerationType.IDENTITY) private long id;
MAPEO	<mapping class="hibernate.relacionesanotaciones.modelo.Persona" /> <mapping class="hibernate.relacionesanotaciones.modelo.Direccion" />	
BIDIRECCIONAL	@Entity public class Pais implements Serializable { @Id @GeneratedValue(strategy = GenerationType.IDENTITY) private int id; @OneToOne(cascade={CascadeType.PERSIST,CascadeType.REMOVE}) private Presidente presidente;	@Entity public class Presidente implements Serializable { @Id @GeneratedValue(strategy = GenerationType.IDENTITY) private int id; @OneToOne private Pais pais;
MAPEO	<mapping class="hibernate.relacionesanotaciones.modelo.Pais" /> <mapping class="hibernate.relacionesanotaciones.modelo.Presidente" />	

OneToMany	CLASE One	CLASE Many

UNIDIRECCIONAL	@Entity
	public class Persona implements Serializable
	{
	@Id
	@GeneratedValue(strategy=GenerationType.IDENTITY)
	private long id;
	@OneToMany(cascade=CascadeType.ALL, fetch=FetchType.EAGER)
	private List<Libro> libros = new ArrayList<Libro>();
	
	public void addLibro(Libro libro)
	{
	this.libros.add(libro);
	}
		@Entity
		public class Libro implements Serializable
		{
		@Id
		@GeneratedValue(strategy=GenerationType.IDENTITY)
		private long id;

MAPEO	<mapping class="hibernate.relaciones.unomuchos.anotaciones.modelo.Libro" /> <mapping class="hibernate.relaciones.unomuchos.anotaciones.modelo.Persona" />	

BIDIRECCIONAL	@Entity
	public class Persona implements Serializable
	{
	@Id
	@GeneratedValue(strategy=GenerationType.IDENTITY)
	private long id;
	private String nombre;
	@OneToMany(cascade=CascadeType.ALL, fetch=FetchType.EAGER, mappedBy="persona")
	private List libros = new ArrayList();
	
	public void addLibro(Libro libro)
	{
	this.libros.add(libro);
	}
		@Entity
		public class Libro implements Serializable
		{
		@Id
		@GeneratedValue(strategy=GenerationType.IDENTITY)
		private long id;
		@ManyToOne
		private Persona persona;

MAPEO	<mapping class="hibernate.relaciones.unomuchos.anotaciones.modelo.Libro" /> <mapping class="hibernate.relaciones.unomuchos.anotaciones.modelo.Persona" />	

Many2One	CLASE Many	CLASE One
UNIDIRECCIONAL	@Entity public class Televidente implements Serializable { @Id @GeneratedValue(strategy = GenerationType.IDENTITY) private long id; @ManyToOne private CadenaTelevisiva cadenaFavorita;	@Entity public class CadenaTelevisiva implements Serializable { @Id @GeneratedValue(strategy=GenerationType.IDENTITY) private long id;
MAPEO	*<mapping class="hibernate.relaciones.muchos.uno.anotaciones.modelo.CadenaTelevisiva" /> <mapping class="hibernate.relaciones.muchos.uno.anotaciones.modelo.Televidente" />*	
BIDIRECCIONAL	La relación muchos a uno bidireccional es igual que la relación uno a muchos bidireccional	
MAPEO		

<table>
<tr><th>ManyToMany</th><th>CLASE Many</th><th>CLASE Many</th></tr>
<tr><td>UNIDIRECCIONAL</td><td>

```
@Entity
public class Estudiante implements Serializable
{
   @Id
   @GeneratedValue(strategy = GenerationType.IDENTITY)
   private long id;
   @ManyToMany(cascade = CascadeType.ALL, fetch = FetchType.EAGER)
   private List<Materia> materias = new ArrayList<Materia>();

public void addMateria(Materia materia)
   {
      this.materias.add(materia);
   }
```

</td><td>

```
@Entity
public class Materia implements Serializable
{
@Id
@GeneratedValue(strategy=GenerationType.IDENTITY)
private long id;
```

</td></tr>
<tr><td>MAPEO</td><td colspan="2">

```
<mapping class="hibernate.relaciones.muchos.muchos.anotaciones.modelo.Estudiante" />
  <mapping class="hibernate.relaciones.muchos.muchos.anotaciones.modelo.Materia" />
```

</td></tr>
<tr><td>BIDIRECCIONAL</td><td>

```
@Entity
public class Estudiante implements Serializable
{
   @Id
   @GeneratedValue(strategy = GenerationType.IDENTITY)
   private long id;
   @ManyToMany(cascade = CascadeType.ALL, fetch = FetchType.EAGER)
   private List<Materia> materias = new ArrayList<Materia>();

public void addMateria(Materia materia)
   {
      this.materias.add(materia);
   }
```

</td><td>

```
@Entity
public class Materia implements Serializable
{
   @Id
   @GeneratedValue(strategy = GenerationType.IDENTITY)
   private long id;
   private String nombre;
   @ManyToMany(mappedBy = "materias")
   private List<Estudiante> estudiantes = new ArrayList<Estudiante>();

public void addEstudiante(Estudiante estudiante)
   {
      this.estudiantes.add(estudiante);
      estudiante.addMateria(this);
   }
```

</td></tr>
<tr><td>MAPEO</td><td colspan="2">

```
<mapping class="hibernate.relaciones.muchos.muchos.anotaciones.modelo.Estudiante" />
  <mapping class="hibernate.relaciones.muchos.muchos.anotaciones.modelo.Materia" />
```

</td></tr>
</table>

ANEXO1 Nombrar tabla y campos

@Entity
@Table(name="contactos")
public class Contacto implements Serializable
{
}
	@Table→ cambia el nombre de la tabla si no toma el de la entidad.

@Column(name="e_mail") private String email;	@Column → cambia el nombre del campo si no toma el de el atributo de la clase.

ANEXO2 Fetch

fetch=FetchType.EAGER	Si la recuperación es "eager" entonces la entidad relacionada se recuperará al mismo tiempo que la entidad dueña
fetch=FetchType.LAZY	Si decidimos que la recuperación sea "lazy" entonces las entidades relacionadas no serán recuperados de la base de datos al momento que se recupera la entidad dueña, sino hasta que se usen estos elementos (siempre y cuando estemos dentro de una transacción)

ANEXO 3 Generación de Claves Primarias

@Id @GeneratedValue(strategy= GenerationType.TABLE) private Long id;	*Simula una secuencia almacenando y actualizando su valor actual en una tabla de base de datos que requiere el uso de bloqueos pesimistas que colocan todas las transacciones en un orden secuencial. Esto ralentiza su aplicación.*
@Id @GeneratedValue(strategy= GenerationType.AUTO) private Long id;	*El GenerationType.AUTO es el tipo de generación por defecto y permite que el proveedor de persistencia elegir la estrategia de generación.* *Si usa Hibernate como su proveedor de persistencia, selecciona una estrategia de generación basada en el dialecto específico de la base de datos.*
@Id @GeneratedValue(strategy= GenerationType.IDENTITY) private Long id;	*Se basa en una columna de base de datos con incremento automático y permite que la base de datos genere un nuevo valor con cada operación de inserción. Desde el punto de vista de la base de datos, esto es muy eficiente porque las columnas de incremento automático están altamente optimizadas y no requiere ninguna declaración adicional.* *Este enfoque tiene un inconveniente importante si usa Hibernate, ya que requiere un valor de clave principal para cada entidad administrada y, por lo tanto, debe realizar la instrucción de inserción de inmediato. Esto evita que utilice diferentes técnicas de optimización como el procesamiento por lotes JDBC.*
@Id @GeneratedValue(strategy= GenerationType.SEQUENCE) private Long id;	*El GenerationType.SEQUENCE utiliza una secuencia de bases de datos para generar valores únicos.* *Requiere sentencias select adicionales para obtener el siguiente valor de una secuencia de base de datos. Pero esto no tiene impacto en el rendimiento para la mayoría de las aplicaciones. Y si su aplicación tiene que persistir una gran cantidad de nuevas entidades, puede usar algunas optimizaciones especificas de hibernate para reducir la cantidad de declaraciones.*

ANEXO 4 JPA Cascade Types

The cascade types supported by the Java Persistence Architecture are as below:

1. CascadeType.PERSIST : cascade type presist means that save() or persist() operations cascade to related entities.
2. CascadeType.MERGE : cascade type merge means that related entities are merged when the owning entity is merged.
3. CascadeType.REFRESH : cascade type refresh does the same thing for the refresh() operation.
4. CascadeType.REMOVE : cascade type remove removes all related entities association with this setting when the owning entity is deleted.
5. CascadeType.DETACH : cascade type detach detaches all related entities if a “manual detach” occurs.
6. CascadeType.ALL : cascade type all is shorthand for all of the above cascade operations.

There is no default cascade type in JPA. By default no operations are cascaded.

Ejemplo:

```
@OneToOne(cascade={CascadeType.PERSIST, CascadeType.REMOVE})
```