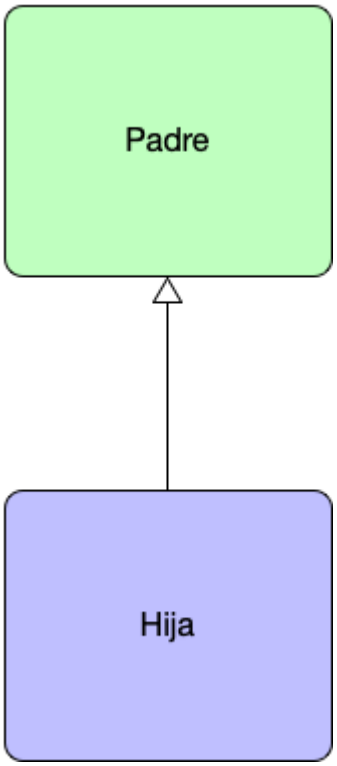
JPA @MappedSuperClass es una anotación que nos puede ser muy útil ya que en algunas ocasiones tenemos entidades de JPA que comparten una serie de atributos entre todas y en vez de repetirlos los podemos diseñar en una clase padre y heredarlos por todas las clases hijas.



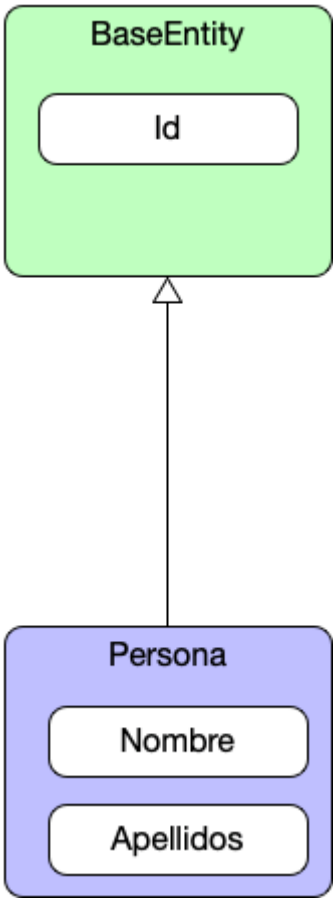
Vamos a ver un ejemplo sencillo de esta anotación . Imaginemonos que tenemos la clase Persona que necesita de un id autoincremental para poder salvarse en la base de datos.

```
package com.arquitecturajava.models;

import jakarta.persistence.Entity;
import jakarta.persistence.FetchType;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.GenerationType;
import jakarta.persistence.Id;
import jakarta.persistence.JoinColumn;
```

```
import jakarta.persistence.ManyToOne;
import jakarta.persistence.Table;

@Entity
@Table(name="personas")
public class Persona  {
        @Id
        @GeneratedValue(strategy = GenerationType.IDENTITY)
        private int id;
        private String nombre;
        private String apellidos;
        private int edad;
        public int getId() {
                return id;
        }
        public void setId(int id) {
                this.id = id;
        }
        public int getEdad() {
                return edad;
        }
        public void setEdad(int edad) {
                this.edad = edad;
        }
        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 Persona(String nombre, String apellidos, int edad) {
                super();
                this.nombre = nombre;
                this.apellidos = apellidos;
                this.edad = edad;
        }
        public Persona() {
                super();
        }
        public Persona(int id) {
                super();
        }
        @Override
        public String toString() {
                return "Persona [id=" + getId() + ", nombre=" + nombre
+ ", apellidos=" + apellidos + "]";
        }
}
```

En principio es una solución correcta pero puede ser que la mayor parte de las Entidades por no decir todas usen un id para almacenarse en la base de datos por lo tanto estaríamos repitiendo los ids en cada una de las entidades .



JPA @MappedSuperClass una solución sencilla

La anotacion de JPA @MappedSuperClass nos permite enfocar de una manera mas directa . Es decir tenemos dos clases una clase padre que contiene los métodos comunes que se suele denominar BaseEntity y una clase hija que extiende de ella y los reutiliza. Vamos a ver la primera clase:

```
package com.arquitecturajava.models2;

import java.io.Serializable;

import jakarta.persistence.GeneratedValue;
import jakarta.persistence.GenerationType;
```

```
import jakarta.persistence.Id;
import jakarta.persistence.MappedSuperclass;

@MappedSuperclass
public class BaseEntity implements Serializable {

	/**
	 *
	 */
	private static final long serialVersionUID = 1L;
	@Id
	@GeneratedValue(strategy = GenerationType.IDENTITY)
	private int id;

	public int getId() {
		return id;
	}

	public void setId(int id) {
		this.id = id;
	}

}
```

La segunda clase simplemente extiende de la primera clase :

```
package com.arquitecturajava.models2;

import jakarta.persistence.Entity;
import jakarta.persistence.FetchType;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.GenerationType;
```

```
import jakarta.persistence.Id;
import jakarta.persistence.JoinColumn;
import jakarta.persistence.ManyToOne;
import jakarta.persistence.Table;

@Entity
@Table(name="personas")
public class Persona extends BaseEntity {

	private String nombre;
	private String apellidos;
	private int edad;
	public int getEdad() {
		return edad;
	}
	public void setEdad(int edad) {
		this.edad = edad;
	}
	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 Persona(String nombre, String apellidos, int edad) {
```

```
                super();
                this.nombre = nombre;
                this.apellidos = apellidos;
                this.edad = edad;
        }
        public Persona() {
                super();
        }
        public Persona(int id) {
                super();
        }
        @Override
        public String toString() {
                return "Persona [id=" + getId() + ", nombre=" + nombre
+ ", apellidos=" + apellidos + "]";
        }
}
```

Esto nos permite simplificar y reducir el código de la Persona sin que nos tengamos que preocupar de los Ids. En otras situaciones se puede usar también para una jerarquía de Herencia compleja.

Otros artículos relacionados

- Entity to DTO y Java 8
- REST HTTP return codes y sus curiosidades
- Visual Studio Code REST Client
- Command Pattern en Java y la gestion de tareas
- ¿Tiene futuro JSF con Jakarta EE?
- JPA