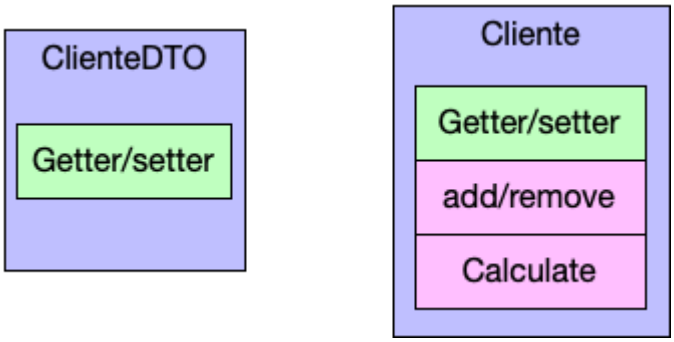
El concepto de Java DTO es uno de los conceptos que utilizamos más en nuestro día a día. Al diseñar arquitecturas REST necesitamos en muchas ocasiones utilizar Data Transfer Objects. ¿Qué diferencia hay entre un DTO y una Entidad u Objeto de negocio? . Normalmente la Entidad esta fuertemente ligada a la persistencia y cuando usamos tecnologías como JPA pues suelen estar cargadas de anotaciones @Entity @Id etc , no solo eso sino que incluyen métodos de negocio y relaciones profundas. Sin embargo los Java DTO son bastantes más simples ya que se componen de propiedades getter/setters y constructores.



Java DTO y Herencia

¿Puede un DTO usar la herencia para obtener propiedades de otro? . Esta es una buena pregunta y la respuesta es simple . Si nos es útil porque no.

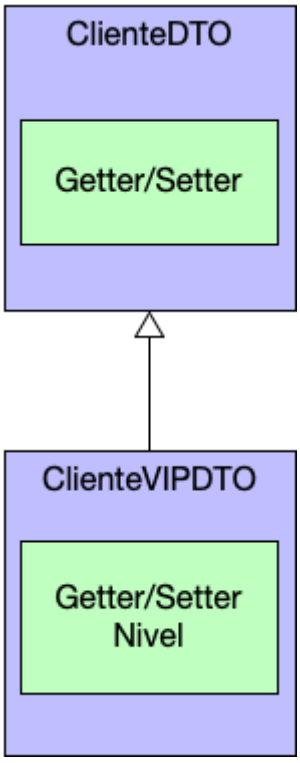
```
package com.aruitecturajava;

public class ClienteDTO {

	private String dni;
	private String nomre;
	public String getDni() {
		return dni;
	}
	public void setDni(String dni) {
		this.dni = dni;
```

```
        }
        public String getNomre() {
                return nomre;
        }
        public void setNomre(String nomre) {
                this.nomre = nomre;
        }
}
```

Los Data Transfer Objects son clases como el resto y pueden usar las capacidades de la programación orientada a objeto como mejor les encaje .



Por lo tanto es posible tener una clase ClienteDTO y heredar ClienteDTOVIP que tiene más propiedades.

```
package com.aruitecturajava;
```

```
public class ClienteDTOVIP extends ClienteDTO {

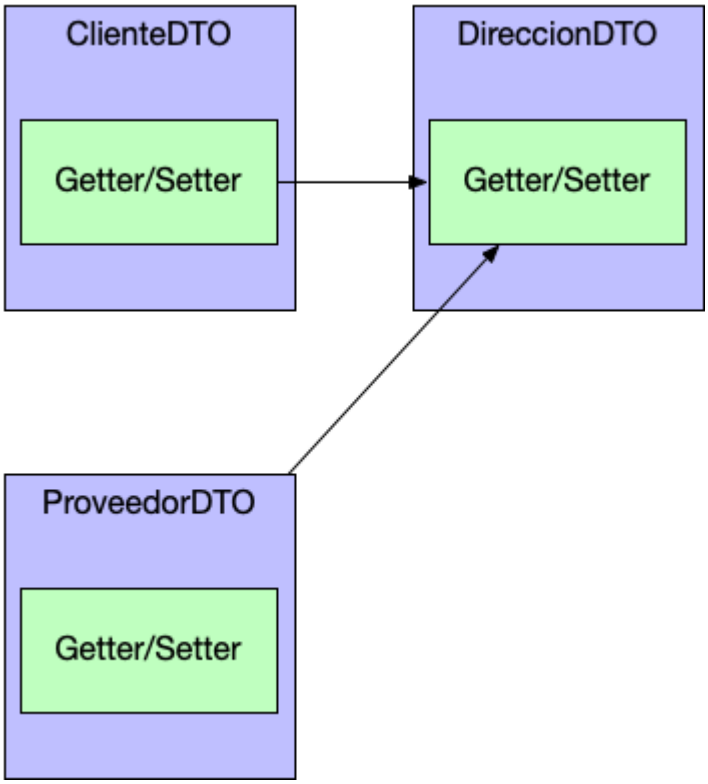
	private int nivel;

	public int getNivel() {
		return nivel;
	}

	public void setNivel(int nivel) {
		this.nivel = nivel;
	}
}
```

Java DTO y Composición

Otra de las opciones posibles es que un DTO contenga dentro de él otro DTO por composición . Hay en mas de una ocasión que esta situación ocurre un ejemplo claro con la clase ClienteDTO puede ser el concepto de DireccionDTO. En la cual el ClienteDTO necesita hacer uso del concepto de DirecciónDTO pero este concepto puede ser compartido también con ProveedorDTO.



Vamos a ver un ejemplo sencillo de código que usa DTO compuestos con delegacion:

```
package com.aruitecturajava;

public class ClienteDTO {

	private String dni;
	private String nomre;
	private DireccionDTO direccion;
	public String getCalle() {
		return direccion.getCalle();
	}
	public void setCalle(String calle) {
		direccion.setCalle(calle);
	}
	public int getNumero() {
```

```
                return direccion.getNumero();
        }
        public void setNumero(int numero) {
                direccion.setNumero(numero);
        }
        public String getDni() {
                return dni;
        }
        public void setDni(String dni) {
                this.dni = dni;
        }
        public String getNomre() {
                return nomre;
        }
        public void setNomre(String nomre) {
                this.nomre = nomre;
        }
}
```

En este caso nos hemos apoyado en el concepto de delegación para construir un DTO más plano pero no es obligatorio. Por lo tanto podemos usar las reglas de programación orientada a objeto en el manejo y construcción de Java DTO si esto nos supone una ventaja.

Otros artículos relacionados

- ¿DTO vs Entity? en Domain Driven Design
- DTO Assembler, un patrón de diseño
- Data Transfer Object (DTO) un concepto clave
- Data Transfer Object