El Principio DRY (Dont Repeat YourSelft) es uno de los principios SOLID y se encarga de evitar la repetición de código . Es quizás el principio más sencillo de entender de todos los principios SOLID. Incluso esta incluido en Eclipse , IntelliJ y Visual Studio Code en varios de los refactorings posibles como Extract Method o Extract Class. Eclipse soporta una operativa muy sencilla en el caso de Extract Method a la hora de aplicar el principio y no repetir código veamos un ejemplo clásico.

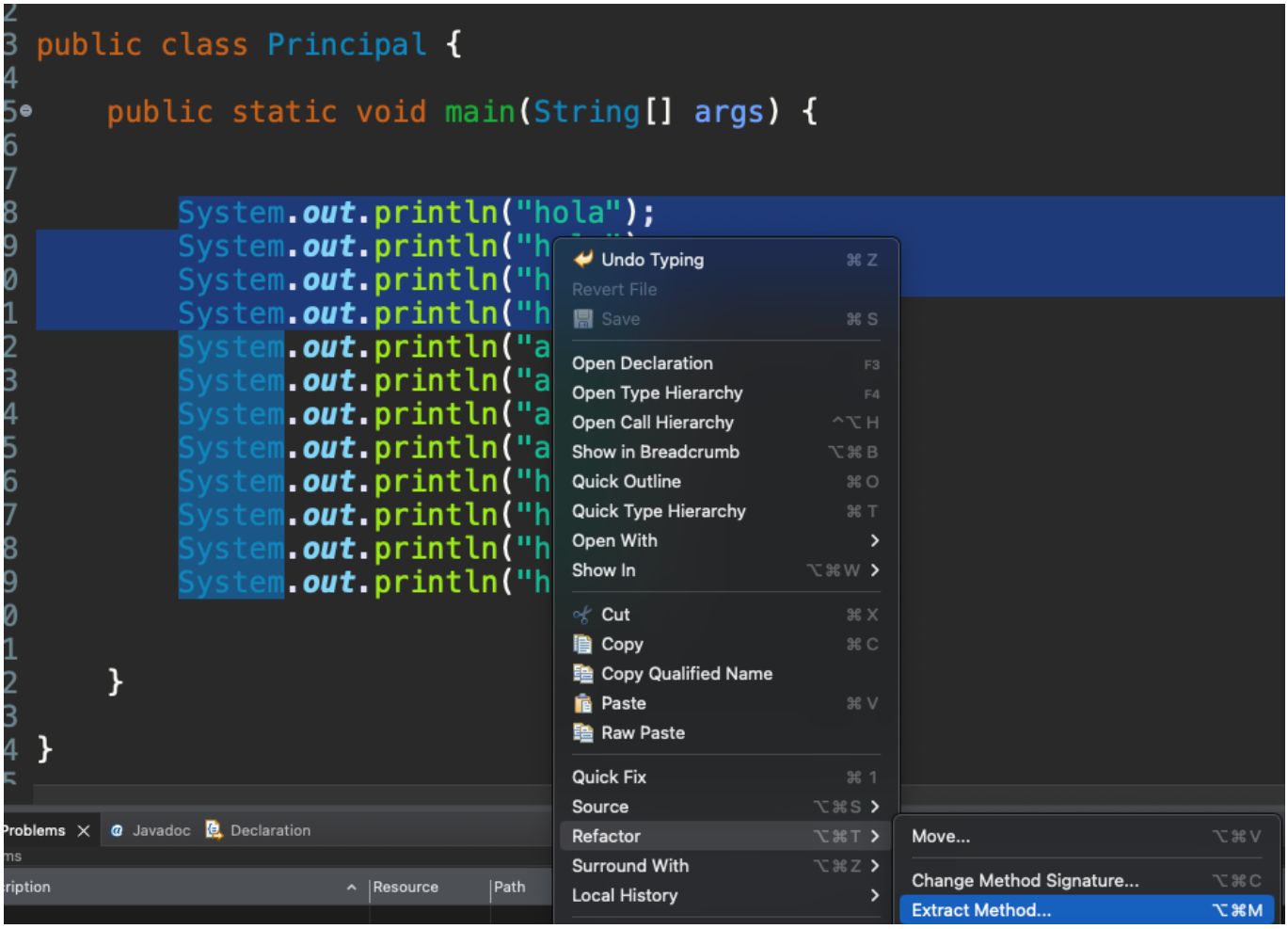
```
public class Principal {

	public static void main(String[] args) {
		System.out.println("hola");
		System.out.println("hola");
		System.out.println("hola");
		System.out.println("hola");
		System.out.println("adios");
		System.out.println("adios");
		System.out.println("adios");
		System.out.println("adios");
		System.out.println("hola");
		System.out.println("hola");
		System.out.println("hola");
		System.out.println("hola");

	}

}
```

Nosotros en este caso tenemos los mensajes de hola muy repetidos y podemos simplemente usar un refactoring y que Eclipse nos genere un método que aplique el principio DRY.



Una vez aplicado el refactoring nos encontramos que el código queda de la siguiente forma.

```
package com.arquitecturajava;

public class Principal {

	public static void main(String[] args) {
		holas();
		System.out.println("adios");
		System.out.println("adios");
		System.out.println("adios");
		System.out.println("adios");
```

```
                holas();

        }

        private static void holas() {
                System.out.println("hola");
                System.out.println("hola");
                System.out.println("hola");
                System.out.println("hola");
        }

}
```

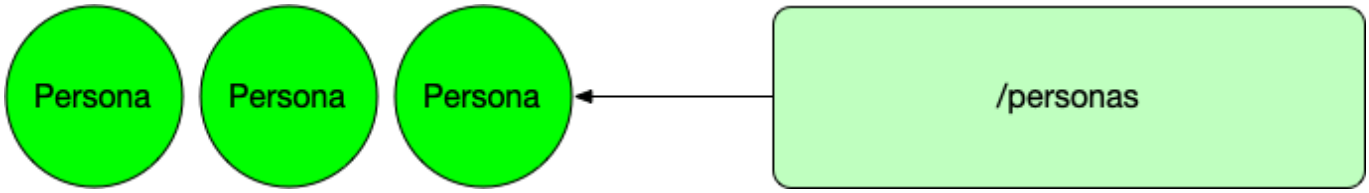
Se ha aplicado correctamente el principio DRY.

Principio DRY ,Encapsulación y reflexiones

Muchas veces aplicamos los principios como leyes absolutas y no lo son. Son guías que nos ayudan a diseñar mejor nuestro software de tal forma que lo que diseñemos sea más flexible o más sólido y se adapte mejor a los cambios.

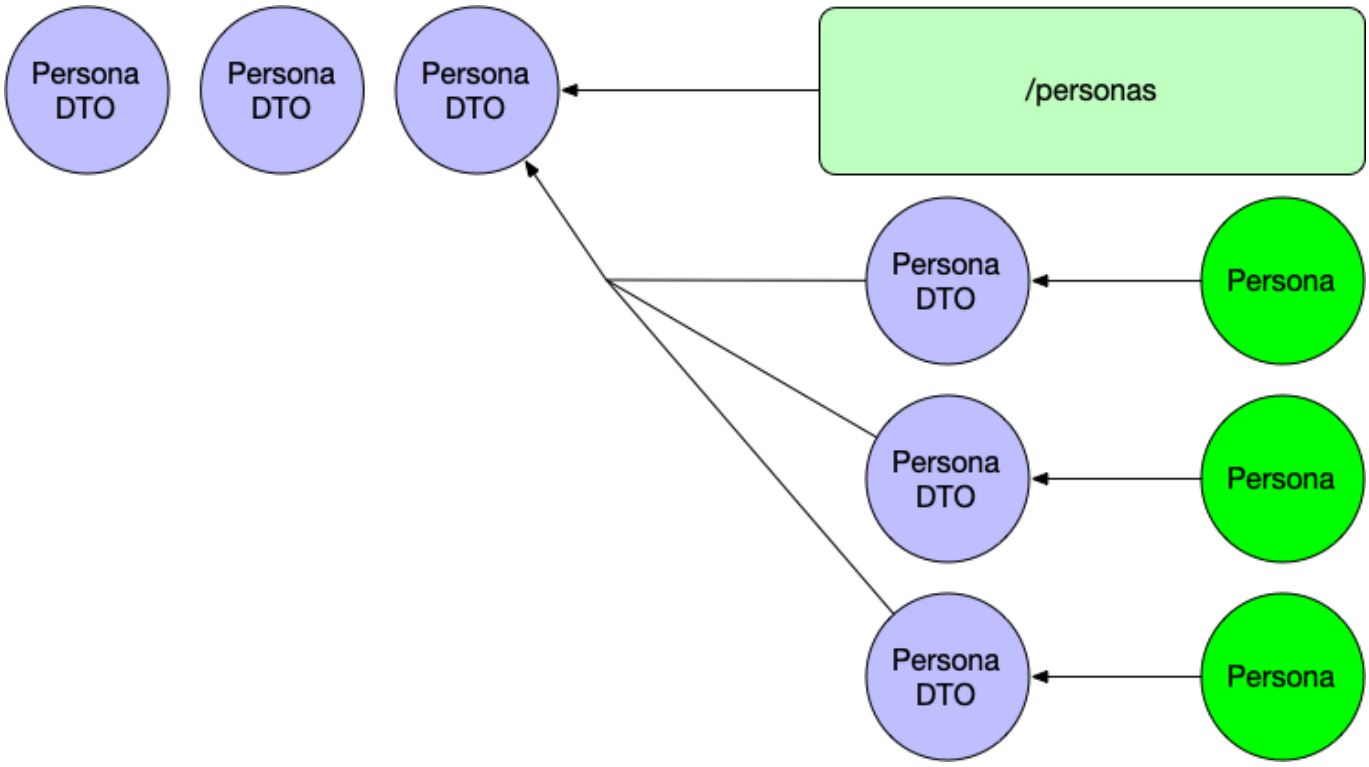
Principio DRY Entities y DTOS

Cuando nosotros disponemos de un servicio REST y este servicio REST necesita publicar datos puede necesitar publicar la información de Facturas. Por lo tanto un desarrollador tiene en su mano simplemente hacer que el servicio publique un listado de Facturas basándonos en la entidad de JPA o similar método de persistencia. Es una solución muy directa a la necesidad que se plantea.



Sin embargo existe otra opción que se usa mucho y es no publicar un listado de Personas sino publicar un listado de PersonaDTO . Es decir convertir primero los objetos Persona a otra clase PersonaDTO que comparte propiedades y constructores .

Eso cambiará el tipo de dato que el servicio REST devuelve haciendo que siempre se devuelva un DTO. Esto contradice el principio DRY ya que parece que estamos repitiendo código ya que a nivel de Entidad es el mismo concepto:



Encapsulación y diseño

En este caso hemos priorizado la encapsulación y aislar el concepto de Persona a nivel de REST del propio principio DRY . Es decir el usuario del API no conoce que exista la clase Persona solo conoce que existe PersonaDTO que para él es como Persona ya que a nivel de JSON el tipo de objeto no es pasado solo sus propiedades. Por lo tanto si por necesidades de negocio tenemos que cambiar algún tipo de propiedad de la clase Persona los clientes REST no tienen porque verse afectados ya que no conocen este concepto. En el momento de transformar adaptaríamos lo que necesitemos. Es aquí donde vemos como priorizar la encapsulación sobre el principio DRY ya que nos aporta más flexibilidad a nivel de REST.

Otros artículos relacionados

- Frameworks ciclo de vida y curiosidades
- DRY vs DAMP dos principios importantes
- El Principio OCP explicado de forma sencilla
- El Principio de Substitución de Liskov
- SOLID