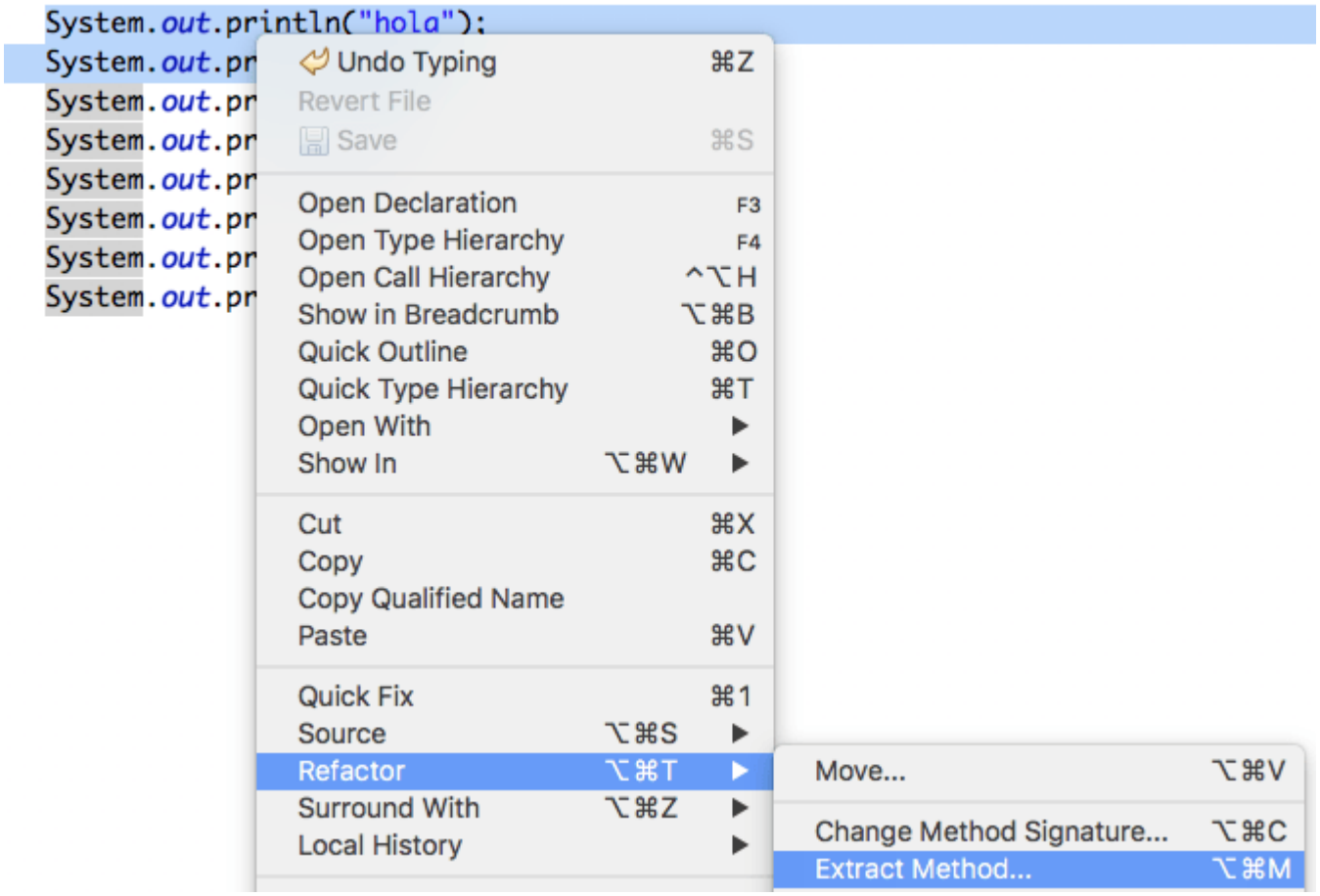
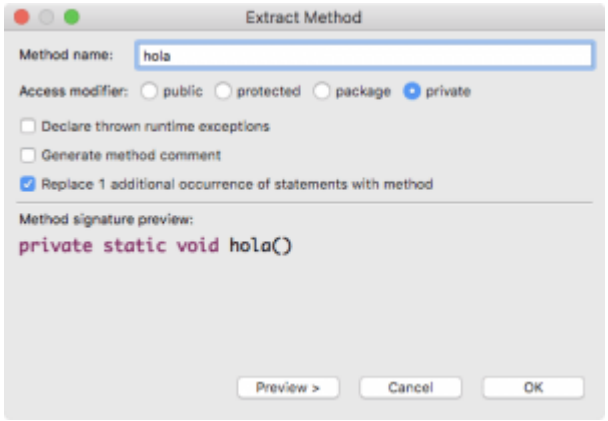
La comparativa entre DRY vs DAMP es bastante clásica a nivel de principios de ingenieria de software. El principio DRY es de sobra conocido por todos (Dont Repeat YourSelf).Hace referencia a que debemos eliminar las repeticiones de código de nuestro programa. Por ejemplo si tenemos este código:

```
package com.arquitecturajava;

public class Principal {

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

}
```

Se puede observar que tiene muchos bloques repetidos si aplicamos el principio DRY podemos generar un par de funciones que se encarguen de eliminar la repetición. Podemos apoyarnos en eclipse en refactor extract method.



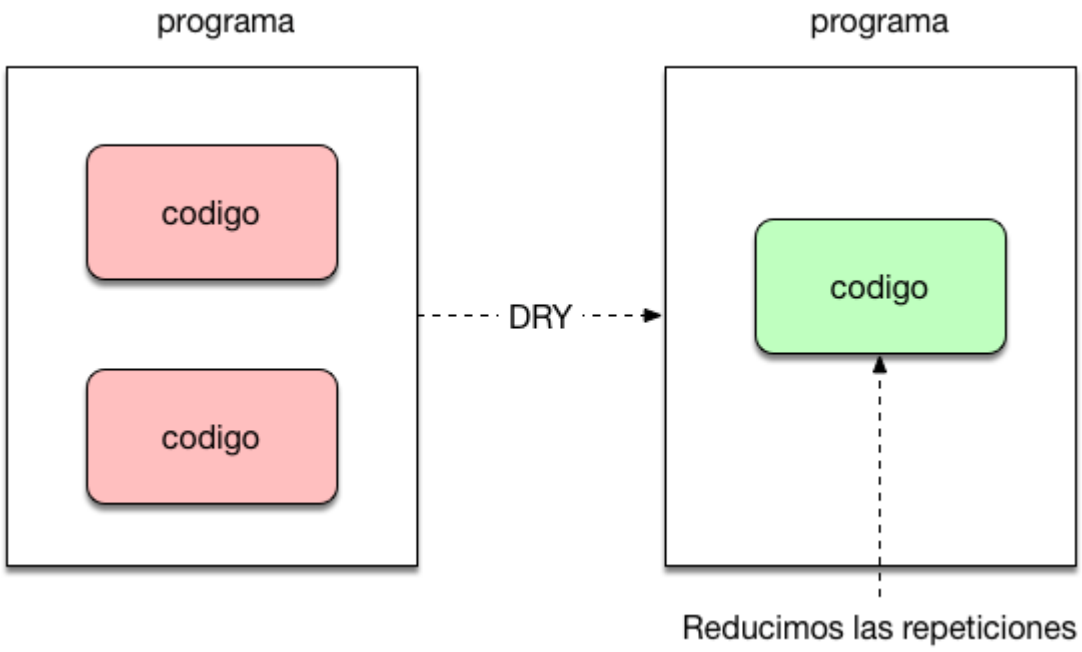

Asignamos el nombre del método:




Eclipse nos refactoriza todo:

```
package com.arquitecturajava;

public class Principal {

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

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

}
```

El Eclipse es lo suficientemente inteligente para darse cuenta que hay varios bloques que se pueden reutilizar , acabamos  de aplicar el principio DRY.




Repetimos la misma operación para el método adios

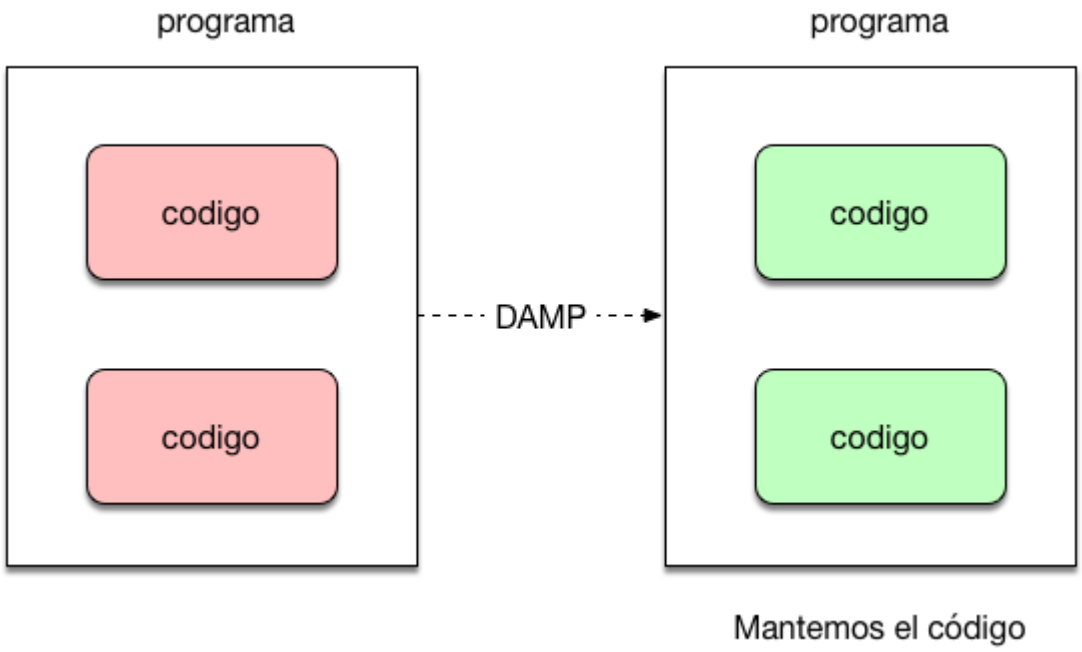
```
package com.arquitecturajava;

public class Principal {

	public static void main(String[] args) {
		hola();
		adios();
		hola();
		adios();
	}

	private static void adios() {
		System.out.println("adios");
```

```
                System.out.println("adios");
        }

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

}
```

Es cierto que podríamos pasar a la función un parámetro y todavía reutilizaríamos más el código ya que el programa se quedaría con una sola función. Es ahí donde seríamos nosotros los que aplicaríamos el principio DRY.

# DRY vs DAMP

El principio DRY se diferencia bastante del principio DAMP (Descriptive And Meaninful Phrases) . Este principio hace referencia a que debemos de ser capaces de construir un código que sea lo suficientemente descriptivo para que le entendamos rápidamente.




Este principio en algunos casos choca con el principio DRY porque este evita la repetición de código pero en algunos casos puede complicar su lectura. Un caso bastante típico es el desarrollo de pruebas unitarias. Echemos un vistazo a lo siguiente:

```
@Test
        public void test_Nueva_Noticia() {

                SimpleDateFormat f = new SimpleDateFormat("yyyy-mm-
dd");

                HashMap<String, String> json = new HashMap<>();
                json.put("titulo", "noticia4");
                json.put("autor", "cecilio");
                json.put("fecha", f.format(new Date()));
                System.out.println(json);
given().contentType(ContentType.JSON).body(json).post("api/noticias");
```

```
                get("/api/noticias").then().body("_embedded.noticias",
IsCollectionWithSize.hasSize(4));


        }

        @Test
        public void test_Nueva_Noticia_Status() {

                SimpleDateFormat f = new SimpleDateFormat("yyyy-mm-
dd");

                HashMap<String, String> json = new HashMap<>();
                json.put("titulo", "noticia4");
                json.put("autor", "cecilio");
                json.put("fecha", f.format(new Date()));
                int
status=given().contentType(ContentType.JSON).body(json).post("api/noti
cias").statusCode();
                assertEquals(201, status);

        }
```

Se trata de dos pruebas unitarias prácticamente iguales. La primera comprueba que un nuevo usuario ha sido insertado. La segunda comprueba que el código de status es un 201. Podríamos haberlas unido en un único test o podríamos haber refactorizado la creación del objeto json que se pasa como parámetro. Pero puede ser que el programador lo vea más claro de esta forma , las pruebas totalmente independientes y con todas sus inicializaciones. En este caso habremos aplicado DAMP en vez de DRY .

Otros artículos relacionados:

1. Java Flyweight pattern y rendimiento
2. Java Composite Pattern y recursividad
3. Utilizando Java Factories y Enums
4. Solid Principles