

# ENTORNOS DE DESARROLLO

PRUEBA ABIERTA IUD4



---

ALUMNO CESUR

24/25

Alejandro Muñoz de la Sierra

PROFESOR

Diego Tinedo Rodríguez

# INTRODUCCION

En el desarrollo de software, uno de los puntos más importantes es hacer que el código sea fácil de leer, mantener y cambiar. Aquí es donde entra la refactorización de código, una técnica que mejora la calidad y estructura del código sin cambiar lo que hace. La refactorización busca hacer que el código sea más eficiente, claro y adaptable a cambios futuros. Este proceso es importante en el ciclo de vida del software, donde los cambios son comunes. En este caso práctico, veremos cuatro patrones básicos de refactorización utilizados en Eclipse: renombrar, extraer interfaz, encapsular campo y extraer método. Cada uno de estos patrones busca mejorar diferentes partes del código, como su modularidad, seguridad, flexibilidad y claridad. Al aplicar estas técnicas, buscamos también mejorar el diseño de las aplicaciones y hacer más fácil su mantenimiento y crecimiento. Así, podemos asegurarnos de que el software se adapta a cambios y evoluciona bien con el tiempo.

# 0 1

## C A M B I O   D E   N O M B R E

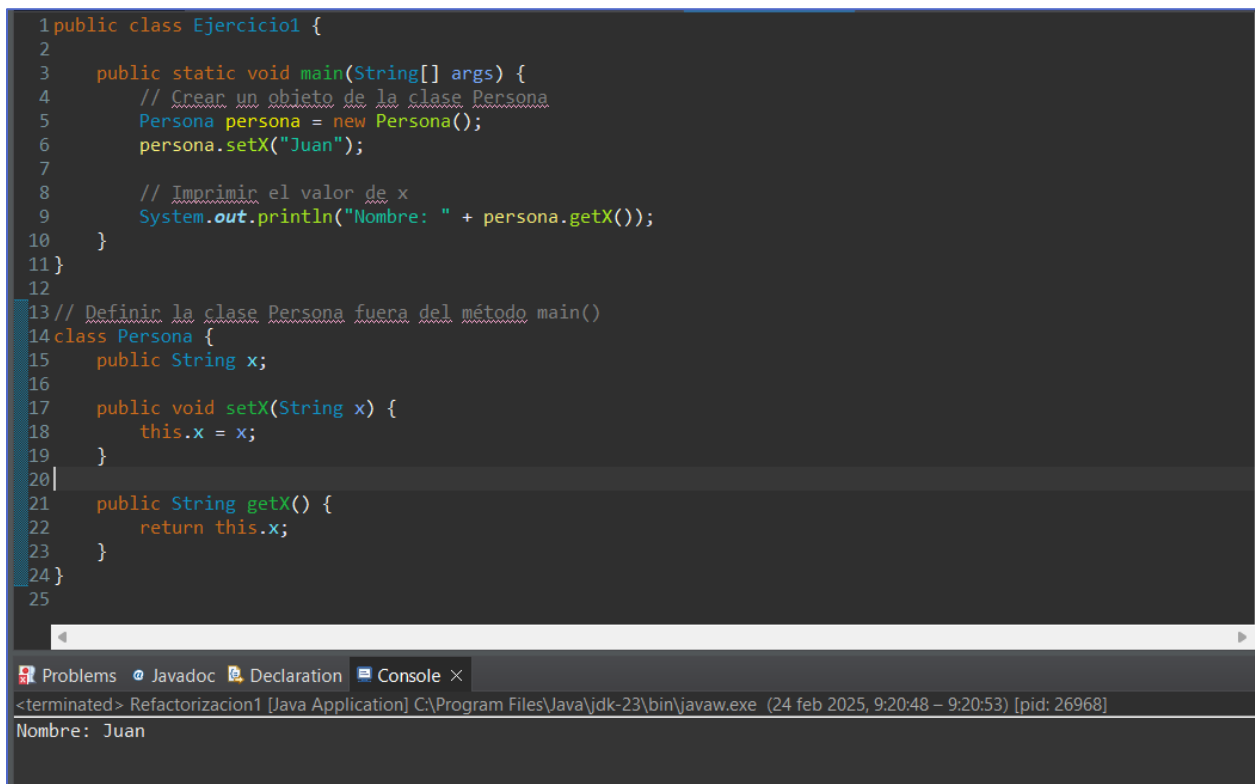
Objetivo: Hacer que el código sea más claro y consistente, asegurando que los nombres de variables, funciones o clases sean significativos y reflejen su función real.

Explicación: Este es un patrón de refactorización que es fácil y útil. Consiste en cambiar nombres que no son claros para que el código sea más fácil de entender para otros programadores.

Ejemplo práctico:

Imaginemos que tenemos la siguiente clase en Java:

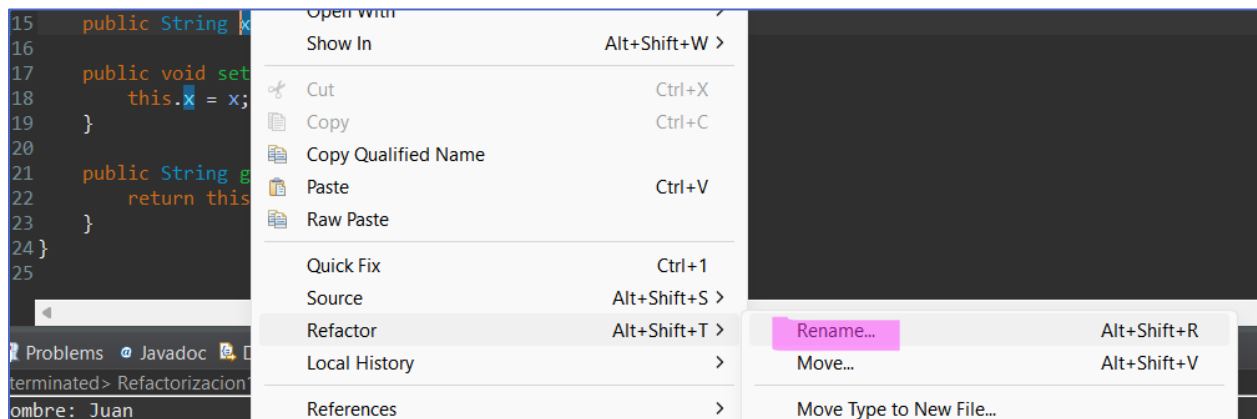
```
1 public class Ejercicio1 {
2
3     public static void main(String[] args) {
4         // Crear un objeto de la clase Persona
5         Persona persona = new Persona();
6         persona.setX("Juan");
7
8         // Imprimir el valor de x
9         System.out.println("Nombre: " + persona.getX());
10    }
11 }
12
13 // Definir la clase Persona fuera del método main()
14 class Persona {
15     public String x;
16
17     public void setX(String x) {
18         this.x = x;
19     }
20
21     public String getX() {
22         return this.x;
23     }
24 }
25
```

The image shows a screenshot of an IDE with a Java code editor and a console window. The code defines a class 'Ejercicio1' with a 'main' method that creates a 'Persona' object, sets its 'x' property to 'Juan', and prints it. The 'Persona' class has 'x' as a public String, and 'setX' and 'getX' methods. The console output at the bottom shows 'Nombre: Juan'.

Aquí, el nombre de la variable `x` no dice nada sobre su uso. Podemos cambiarlo a un nombre más descriptivo como `nombre`.

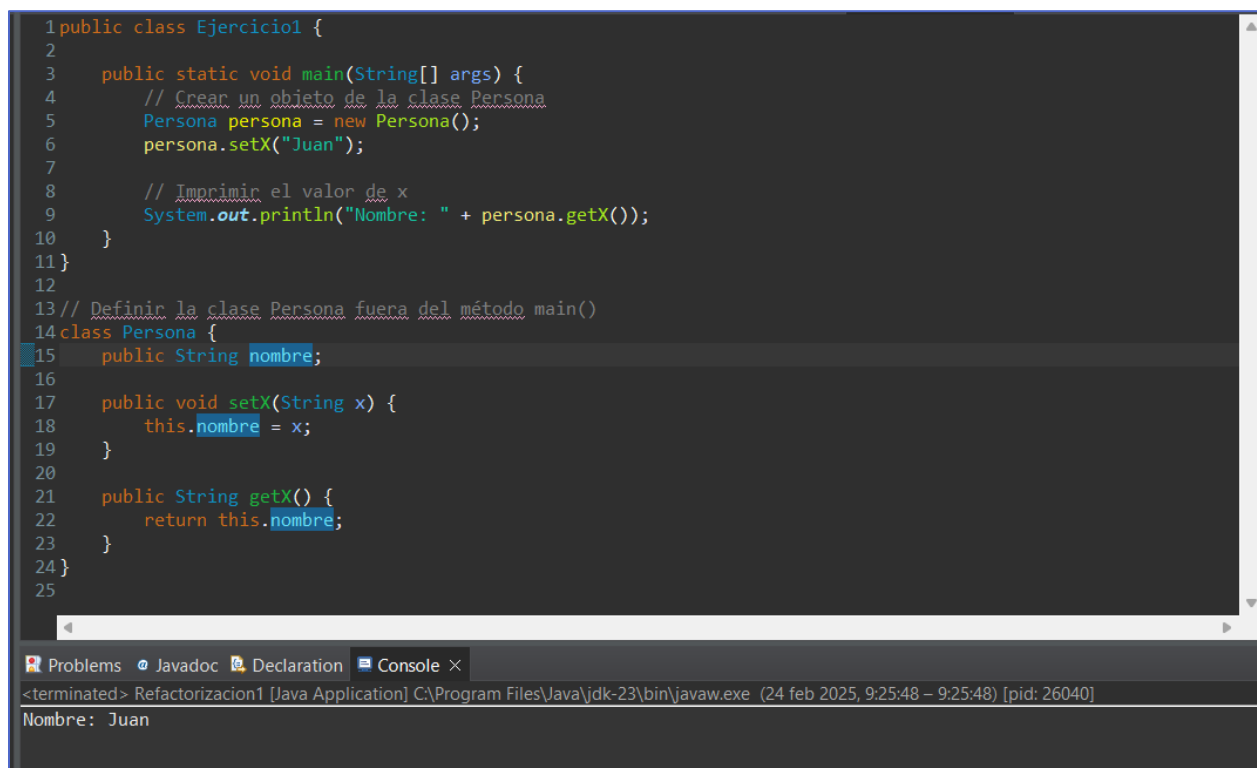
Cómo hacerlo en Eclipse:

1. Haz clic derecho sobre la variable que quieres renombrar.
2. Selecciona Refactor > Rename.
3. Escribe el nuevo nombre y confirma el cambio.



Beneficio: Hacer estos cambios hace que el código sea más claro y más fácil de mantener.

Refactorización: Observamos que obtenemos el mismo resultado al ejecutar el código.



## 02

# EXTRAER INTERFAZ

Objetivo: Hacer el código más modular y flexible creando una interfaz común para varias clases.

Explicación: Si varias clases tienen métodos similares, es útil sacar esos métodos a una interfaz. Esto permite mayor flexibilidad y hace más fácil agregar nuevas clases en el futuro.

Ejemplo práctico:

Supongamos que tenemos una clase `Coche` con los métodos `acelerar()` y `frenar()`:

```
1 public class Ejercicio2 {
2
3     public static void main(String[] args) {
4         // Crear un objeto de la clase Coche
5         Coche miCoche = new Coche();
6
7         // Llamar a los métodos
8         miCoche.acelerar();
9         miCoche.frenar();
10    }
11
12    public static class Coche {
13        public void acelerar() {
14            System.out.println("Acelerando el coche");
15        }
16
17        public void frenar() {
18            System.out.println("Frenando el coche");
19        }
20    }
21 }
22 }
```

Problems Javadoc Declaration Console ×

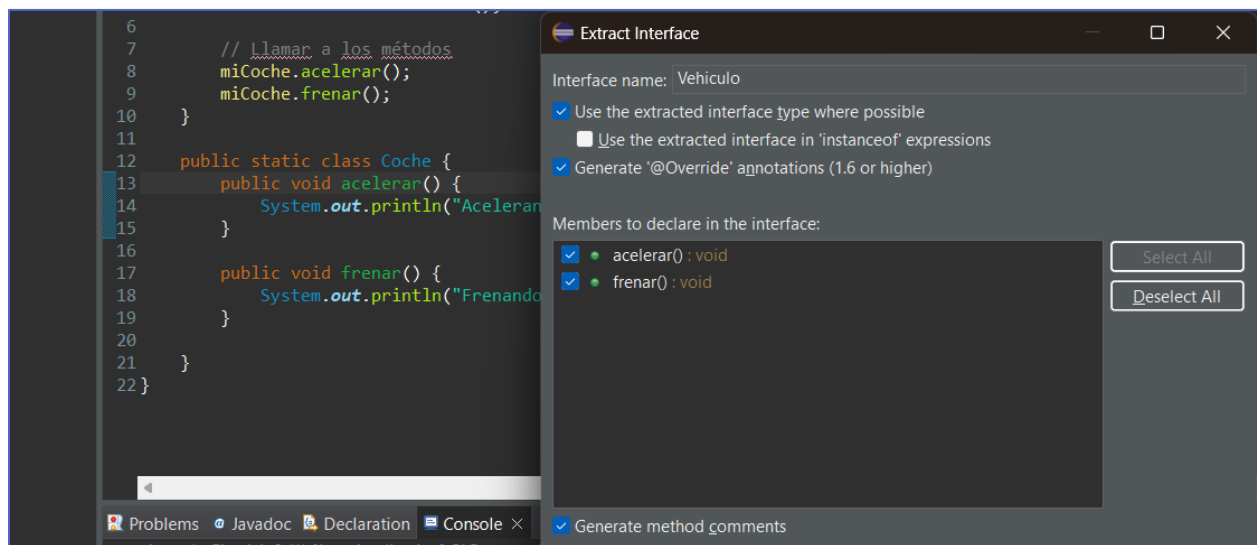
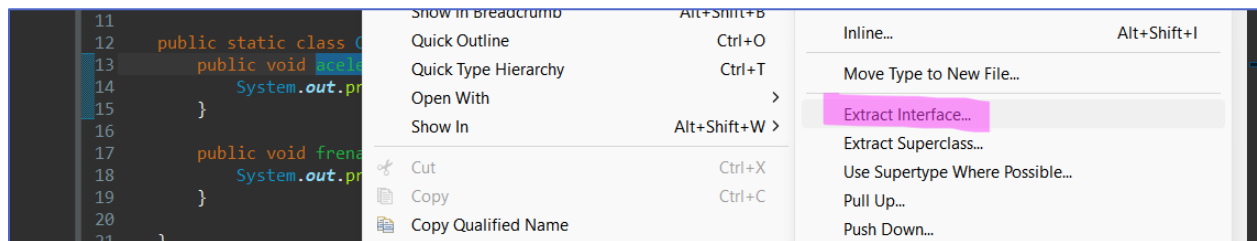
<terminated> Ejercicio2 (1) [Java Application] C:\Program Files\Java\jdk-23\bin\javaw.exe (24 feb 2025, 9:34:11 – 9:34:11) [pid: 10172]

Acelerando el coche  
Frenando el coche

Si queremos que otras clases, como `Bicicleta`, también usen estos métodos, podemos extraerlos a una interfaz llamada `Vehiculo`.

Cómo hacerlo en Eclipse:

1. Haz clic derecho sobre los métodos `acelerar()` y `frenar()` en la clase `Coche`.
2. Selecciona Refactor > Extract Interface.
3. Escribe un nombre para la nueva interfaz (en este caso, `Vehiculo`) y escoge los métodos a extraer.



Beneficio: Ahora, cualquier otra clase que represente un vehículo puede implementar esta interfaz sin depender directamente de `Coche`. Esto mejora la organización del código y facilita su reutilización.

Refactorización: Observamos que obtenemos el mismo resultado al ejecutar el código.

```
1|
2 public interface Vehiculo {
3
4     void acelerar();
5
6     void frenar();
7
8 }
```

```
1 public class Ejercicio2 {
2
3     public static void main(String[] args) {
4         // Crear un objeto de la clase Coche
5         Vehiculo miCoche = new Coche();
6
7         // Llamar a los métodos
8         miCoche.acelerar();
9         miCoche.frenar();
10    }
11
12    public static class Coche implements Vehiculo {
13        @Override
14        public void acelerar() {
15            System.out.println("Acelerando el coche");
16        }
17
18        @Override
19        public void frenar() {
20            System.out.println("Frenando el coche");
21        }
22    }
23 }
24 }
```

Problems Javadoc Declaration Console ×

<terminated> Ejercicio2 (1) [Java Application] C:\Program Files\Java\jdk-23\bin\javaw.exe (24 feb 2025, 9:42:43 – 9:42:43) [pi  
Acelerando el coche  
Frenando el coche

# 03

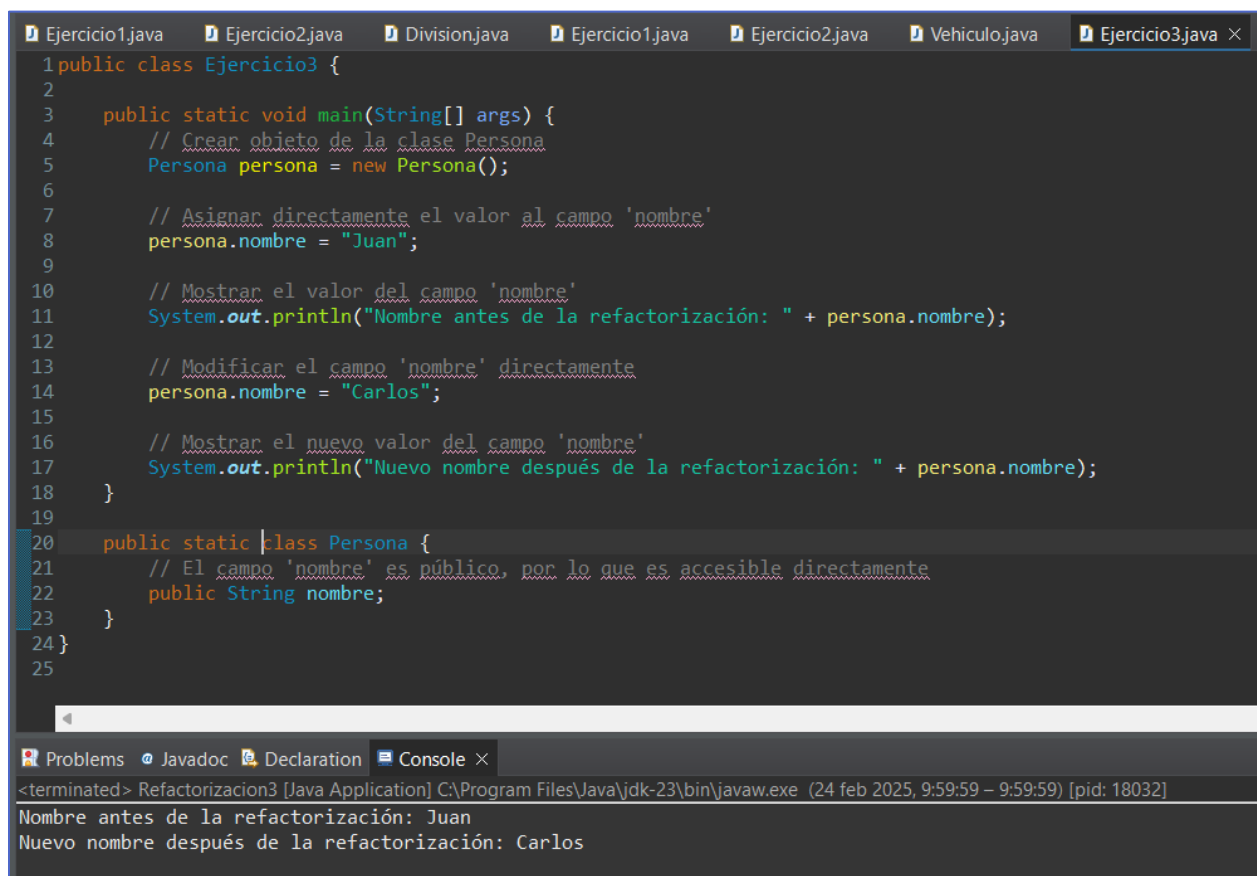
## ENCAPSULAR CAMPO

Objetivo: Mejorar la seguridad y el diseño del código limitando el acceso directo a los atributos de una clase.

Explicación: En programación orientada a objetos, es bueno hacer que los atributos de una clase sean privados y usar métodos públicos (getters y setters) para acceder y cambiar sus valores. Esto previene cambios no deseados y permite un mejor control de los datos.

Ejemplo práctico:

En el siguiente código, el atributo `nombre` es público, permitiendo su modificación desde cualquier parte del programa:



```
1 public class Ejercicio3 {
2
3     public static void main(String[] args) {
4         // Crear objeto de la clase Persona
5         Persona persona = new Persona();
6
7         // Asignar directamente el valor al campo 'nombre'
8         persona.nombre = "Juan";
9
10        // Mostrar el valor del campo 'nombre'
11        System.out.println("Nombre antes de la refactorización: " + persona.nombre);
12
13        // Modificar el campo 'nombre' directamente
14        persona.nombre = "Carlos";
15
16        // Mostrar el nuevo valor del campo 'nombre'
17        System.out.println("Nuevo nombre después de la refactorización: " + persona.nombre);
18    }
19
20    public static class Persona {
21        // El campo 'nombre' es público, por lo que es accesible directamente
22        public String nombre;
23    }
24 }
25
```

Problems Javadoc Declaration Console ×

<terminated> Refactorizacion3 [Java Application] C:\Program Files\Java\jdk-23\bin\javaw.exe (24 feb 2025, 9:59:59 – 9:59:59) [pid: 18032]

Nombre antes de la refactorización: Juan

Nuevo nombre después de la refactorización: Carlos



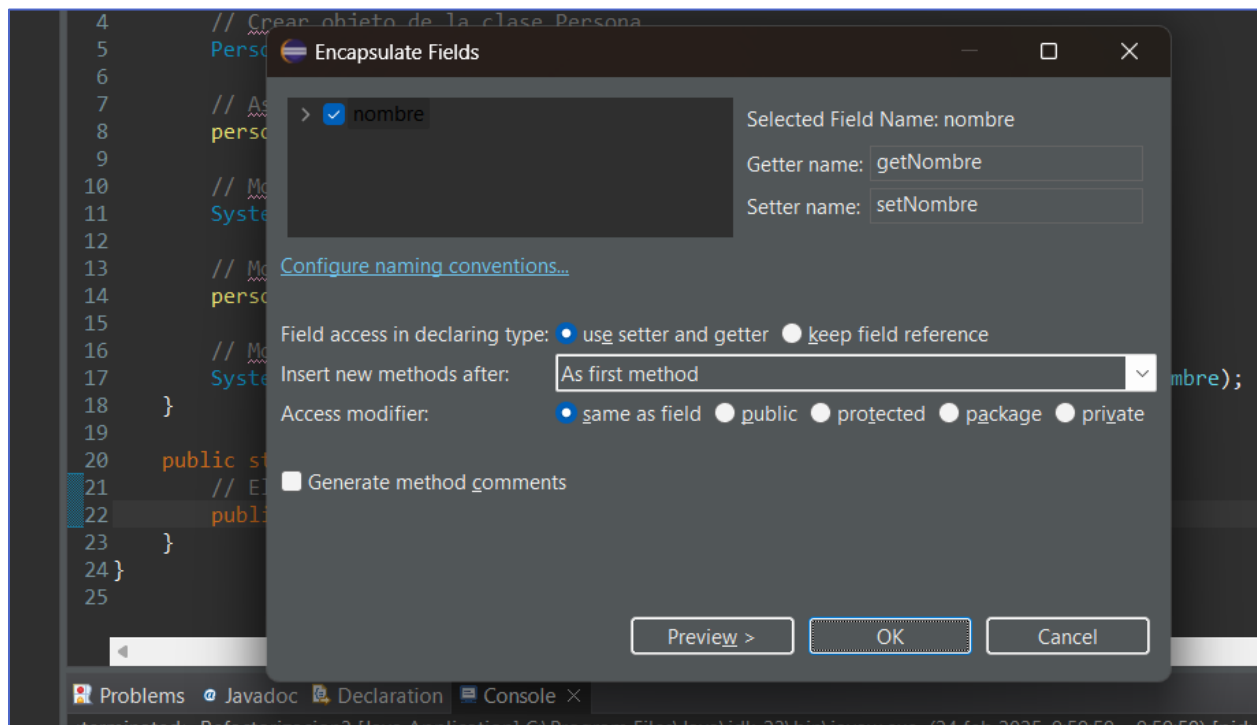
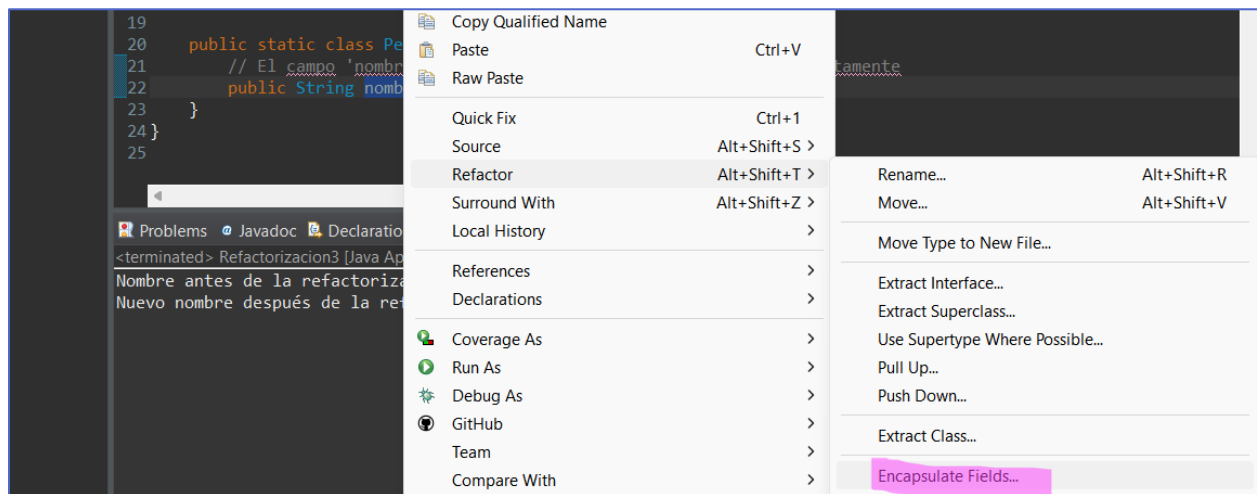
Hacemos el campo privado y añadimos métodos de acceso.

Cómo hacerlo en Eclipse:

Haz clic derecho en el campo nombre.

Elige Refactor > Encapsulate Field.

Eclipse hará automáticamente los métodos `getNombre()` y `setNombre()`, y hará que el atributo sea privado.



Beneficio: Esto impide que otros objetos accedan al campo directamente, mejorando la seguridad y mantenimiento del código.

Refactorización: Observamos que obtenemos el mismo resultado al ejecutar el código.

```
1 public class Ejercicio3 {
2
3     public static void main(String[] args) {
4         // Crear objeto de la clase Persona
5         Persona persona = new Persona();
6
7         // Asignar directamente el valor al campo 'nombre'
8         persona.setNombre("Juan");
9
10        // Mostrar el valor del campo 'nombre'
11        System.out.println("Nombre antes de la refactorización: " + persona.getNombre());
12
13        // Modificar el campo 'nombre' directamente
14        persona.setNombre("Carlos");
15
16        // Mostrar el nuevo valor del campo 'nombre'
17        System.out.println("Nuevo nombre después de la refactorización: " + persona.getNombre());
18    }
19
20    public static class Persona {
21        // El campo 'nombre' es público, por lo que es accesible directamente
22        private String nombre;
23
24        public String getNombre() {
25            return nombre;
26        }
27
28        public void setNombre(String nombre) {
29            this.nombre = nombre;
30        }
31    }
32 }
33
```

Problems Javadoc Declaration Console ×

<terminated> Refactorizacion3 [Java Application] C:\Program Files\Java\jdk-23\bin\javaw.exe (24 feb 2025, 10:05:12 – 10:05:12) [pid: 26940]

Nombre antes de la refactorización: Juan  
Nuevo nombre después de la refactorización: Carlos

## 04

# EXTRAER MÉTODO

Objetivo: Hacer el código más legible y mantenible dividiendo código en métodos separados.

Explicación: Cuando hay trozos de código repetidos o que realizan tareas específicas en un método más grande, es bueno extraerlos a un nuevo método con un nombre claro. Esto ayuda a reutilizarlos y entender mejor el código.

Ejemplo:

Veamos este código, donde hay varias operaciones dentro de un solo método:

```
1 public class Ejercicio4 {
2
3     public static void main(String[] args) {
4         // Crear objeto de la clase Calculadora
5         Calculadora calculadora = new Calculadora();
6
7         // Llamar al método refactorizado
8         calculadora.realizarOperacion();
9     }
10
11     public static class Calculadora {
12         public void realizarOperacion() {
13             int resultado1 = 5 * 2;
14             int resultado2 = 10 / 2;
15             int resultado3 = resultado1 + resultado2;
16             System.out.println("Resultado: " + resultado3);
17         }
18     }
19
20 }
21
```

Problems Javadoc Declaration Console ×

<terminated> Ejercicio4 (1) [Java Application] C:\Program Files\Java\jdk-23\bin\javaw.exe (24 feb 2025, 10:12:19 – 10:12:19)

Resultado: 15

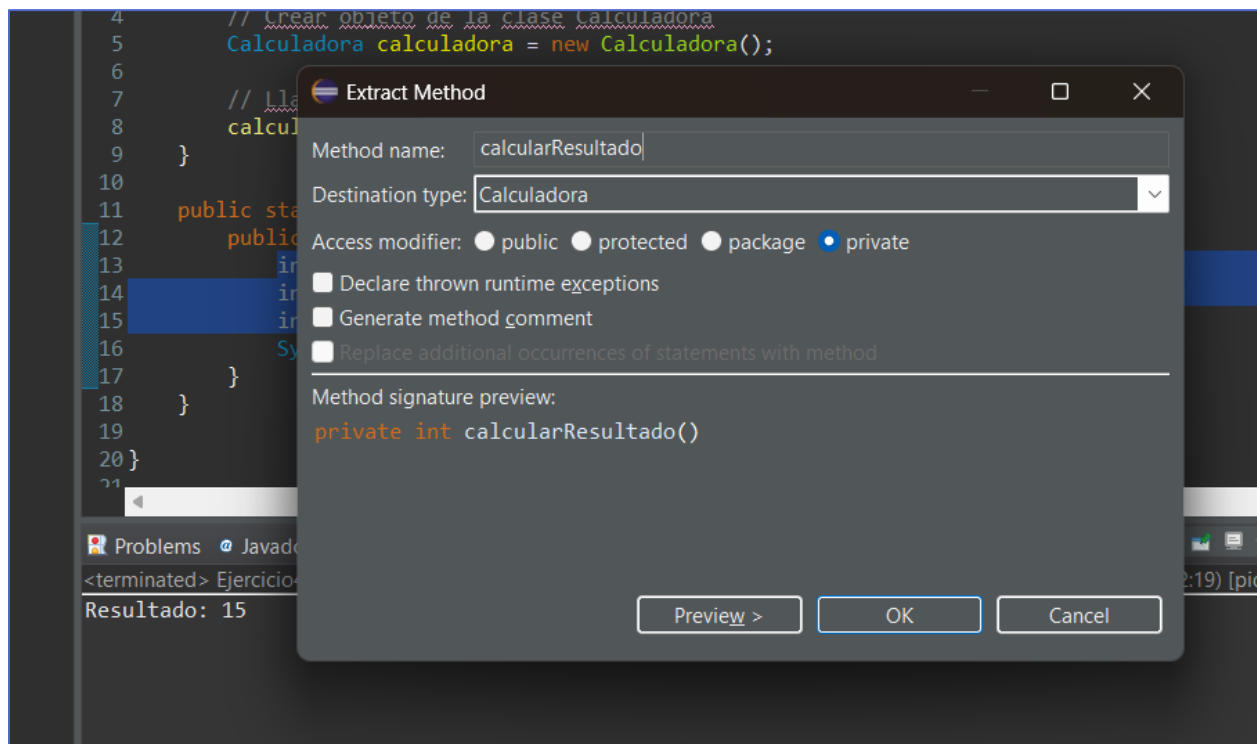
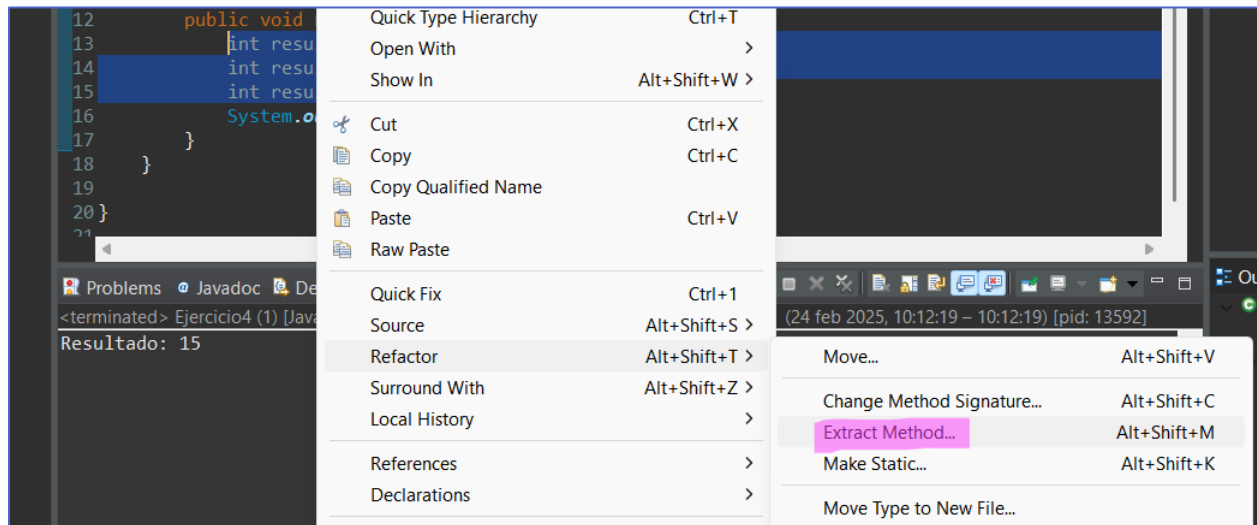
Aquí podemos extraer la lógica en un método distinto llamado `calcularResultado()`, haciendo el código más limpio.

Cómo hacerlo en Eclipse:

Selecciona el bloque de código a extraer.

Haz clic derecho y elige Refactor > Extract Method.

Pon un nombre claro para el nuevo método (aquí, calcularResultado()).



Beneficio: Ahora, la operación de cálculo está separada claramente, lo que facilita su comprensión y cambio sin tocar otras partes.

Refactorización: Observamos que obtenemos el mismo resultado al ejecutar el código.

```
1 public class Ejercicio4 {
2
3     public static void main(String[] args) {
4         // Crear objeto de la clase Calculadora
5         Calculadora calculadora = new Calculadora();
6
7         // Llamar al método refactorizado
8         calculadora.realizarOperacion();
9     }
10
11     public static class Calculadora {
12         public void realizarOperacion() {
13             int resultado3 = calcularResultado();
14             System.out.println("Resultado: " + resultado3);
15         }
16
17         private int calcularResultado() {
18             int resultado1 = 5 * 2;
19             int resultado2 = 10 / 2;
20             int resultado3 = resultado1 + resultado2;
21             return resultado3;
22         }
23     }
24
25 }
```

Problems Javadoc Declaration Console ×

<terminated> Ejercicio4 (1) [Java Application] C:\Program Files\Java\jdk-23\bin\javaw.exe (24 feb 2025, 10:19:55 – 10:19:55) [pid: 7688]

Resultado: 15

## CONCLUSIONES

En este caso práctico, hemos utilizado diferentes técnicas de refactorización que nos han mostrado cómo un pequeño cambio en la estructura puede tener un gran efecto en la calidad del software. Desde cambiar un nombre para que el código sea más claro hasta extraer métodos para organizar y simplificar la lógica del programa, cada patrón de refactorización tiene beneficios claros. Hacer bien la refactorización mejora no solo la legibilidad y mantenimiento del código, sino que también ayuda a implementar nuevas funciones y cambios en el futuro. Usar herramientas como Eclipse para estos cambios acelera el proceso, permitiéndonos ser más productivos y tener más control sobre el software que hacemos. En resumen, entender estos patrones de refactorización es crucial para cualquier desarrollador que quiera hacer código limpio, eficiente y listo para cambios, adaptándose a nuevas necesidades del proyecto. La refactorización no solo mejora el software, sino que también ayuda a una mejor experiencia de desarrollo y mantenimiento a largo plazo.

## REFERENCIAS

[https://help.eclipse.org/latest/index.jsp?topic=%2Forg.eclipse.cdt.doc.user%2Ftasks%2Fcdt\\_t\\_rename.htm](https://help.eclipse.org/latest/index.jsp?topic=%2Forg.eclipse.cdt.doc.user%2Ftasks%2Fcdt_t_rename.htm)

<https://www.youtube.com/watch?v=JZn4hIMvP60>

<https://www.oreilly.com/library/view/eclipse-cookbook/0596007108/ch04s04.html#:~:text=To%20extract%20an%20interface%20from%20the%20Interfaces%20class%2C%20right%2Dclick,name%20NewInterface%20%2C%20and%20click%20OK.>

<https://recacha.wordpress.com/2012/08/30/refactoring-en-eclipse-9-self-encapsulate-field/>

<https://help.eclipse.org/latest/index.jsp?topic=%2Forg.eclipse.jdt.doc.user%2FgettingStarted%2Fqs-ExtractMethod.htm>

<https://www.baeldung.com/eclipse-refactoring>

<https://www.youtube.com/watch?v=ugRw4YHpKq4>