

Entornos de Desarrollo

Refactorización

UD2 Actividad 5

INDICE

[Renombrar \(Rename\)](#)

[Extraer Método \(Extract Method\)](#)

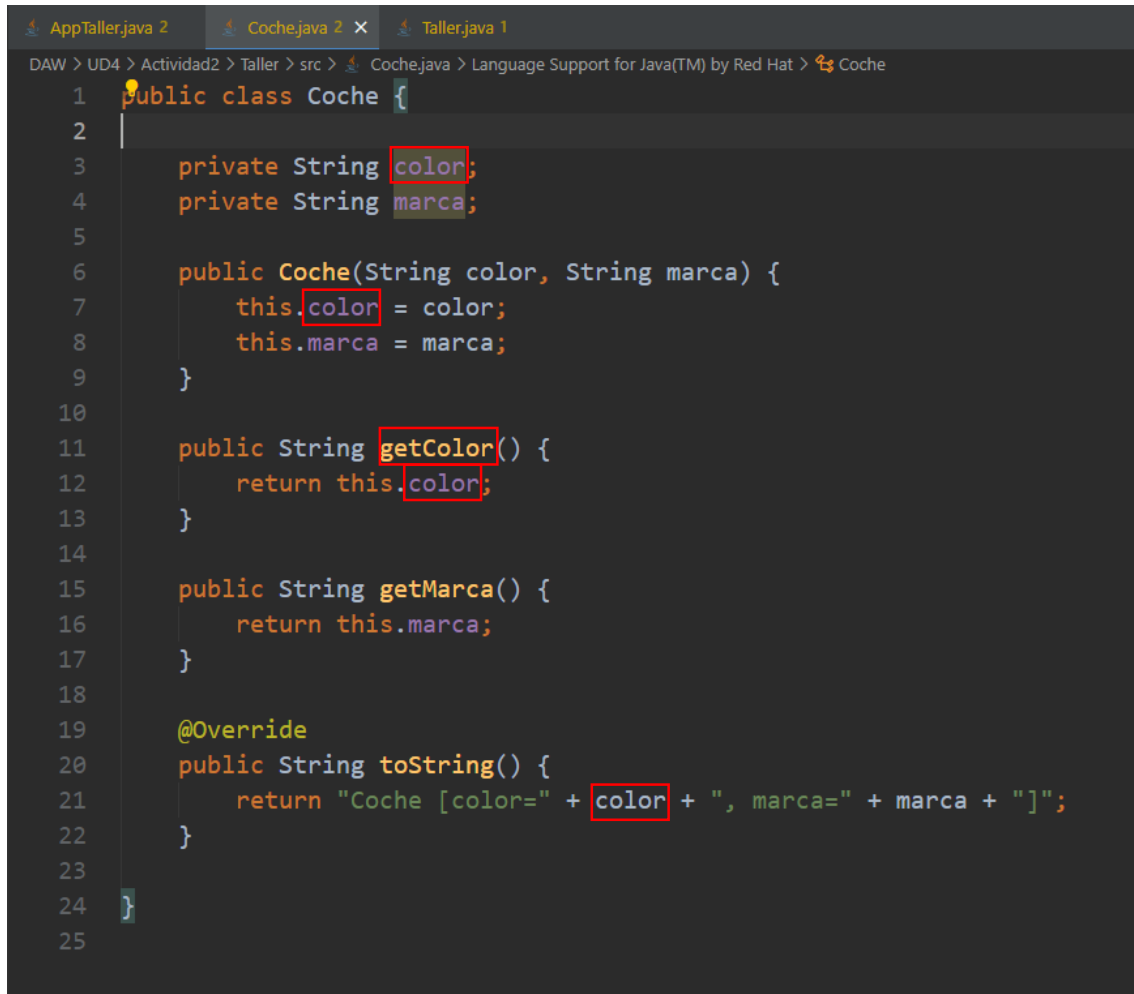
[Mover Archivos o Clases](#)

[Eliminar Código Muerto \(Safety Delete\)](#)

[Refactorización Manual de Métodos y Parámetros](#)

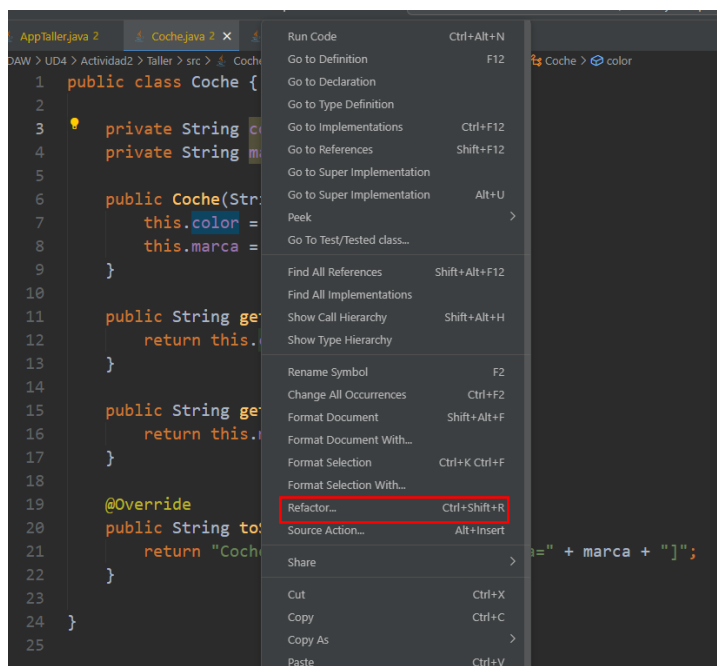
Renombrar (Rename)

Cambia el nombre de un archivo, clase, método o variable.



```
1 public class Coche {
2
3     private String color;
4     private String marca;
5
6     public Coche(String color, String marca) {
7         this.color = color;
8         this.marca = marca;
9     }
10
11     public String getColor() {
12         return this.color;
13     }
14
15     public String getMarca() {
16         return this.marca;
17     }
18
19     @Override
20     public String toString() {
21         return "Coche [color=" + color + ", marca=" + marca + "];";
22     }
23
24 }
25
```

Encontramos la variable que queremos renombrar y lo seleccionamos, en mi caso es **color**, podemos hacerlo con **(F2)** o también con click derecho sobre la variable y encontramos "Rename Symbol"



```

1 public class Coche {
2
3     private String color;
4     private String w;
5
6     public Coche(String color, String marca) {
7         this.color = color;
8         this.marca = marca;
9     }

```

Sale una ventana donde insertamos el nueva variable

```

1 public class Coche {
2
3     private String color;
4     private String marca;
5
6     public Coche(String color, String marca) {
7         this.color = color;
8         this.marca = marca;
9     }
10
11     public String getColor() {
12         return this.color;
13     }
14
15     public String getMarca() {
16         return this.marca;
17     }
18
19     @Override
20     public String toString() {
21         return "Coche [color=" + color + ", marca=" + marca + "]";
22     }
23
24 }

```

Despues cambia todos los variables con nombre que tenían antes (**Coche -> w**)

Extraer Método (Extract Method)

Si tienes un método o un bloque de código largo, puedes extraerlo en un nuevo método para hacerlo más legible.

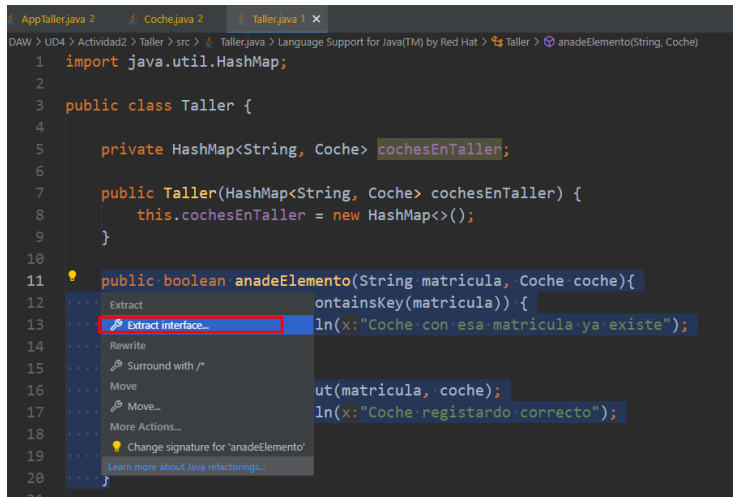
```

1 import java.util.HashMap;
2
3 public class Taller {
4
5     private HashMap<String, Coche> cochesEnTaller;
6
7     public Taller(HashMap<String, Coche> cochesEnTaller) {
8         this.cochesEnTaller = new HashMap<>();
9     }
10
11     public boolean anadeElemento(String matricula, Coche coche){
12         if (cochesEnTaller.containsKey(matricula)) {
13             System.out.println(x:"Coche con esa matricula ya existe");
14             return false;
15         }else{
16             cochesEnTaller.put(matricula, coche);
17             System.out.println(x:"Coche registrando correcto");
18             return true;
19         }
20     }
21
22     public void eliminaElemento(String matricula){
23         if (cochesEnTaller.containsKey(matricula)) {
24             cochesEnTaller.remove(matricula);
25             System.out.println(x:"Coche eliminado");
26         }else{
27             System.out.println(x:"No existe coche con este matricula");
28         }
29     }
30
31     public void visualizaMatriculas(){
32         if (cochesEnTaller.isEmpty()) {
33             System.out.println(x:"No hay coches en taller");
34         }
35     }
36 }

```

Seleccionamos

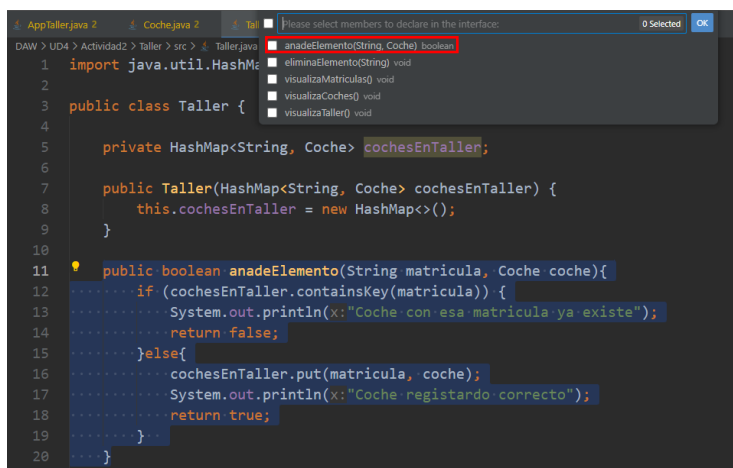
el método y con click derecho elegimos "Refactor..." o Ctrl+Shift+R.



```
1 import java.util.HashMap;
2
3 public class Taller {
4
5     private HashMap<String, Coche> cochesEnTaller;
6
7     public Taller(HashMap<String, Coche> cochesEnTaller) {
8         this.cochesEnTaller = new HashMap<>();
9     }
10
11     public boolean anadeElemento(String matricula, Coche coche){
12         if (cochesEnTaller.containsKey(matricula)) {
13             ln(x:"Coche con esa matricula ya existe");
14         }
15         cochesEnTaller.put(matricula, coche);
16         ln(x:"Coche registrado correcto");
17     }
18 }
19
20
```

The 'Extract' option is highlighted in the refactoring menu.

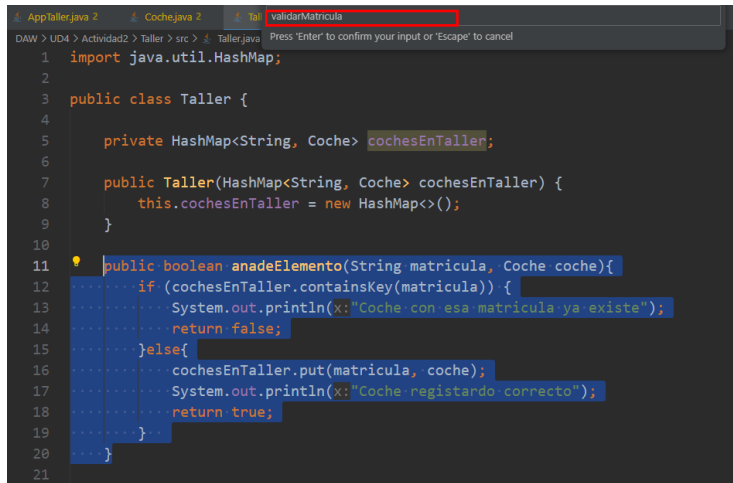
Aquí tenemos tres posibilidades de refactorización extract, rewrite o move. Nuestro caso ahora es “Extract”.



```
1 import java.util.HashMap;
2
3 public class Taller {
4
5     private HashMap<String, Coche> cochesEnTaller;
6
7     public Taller(HashMap<String, Coche> cochesEnTaller) {
8         this.cochesEnTaller = new HashMap<>();
9     }
10
11     public boolean anadeElemento(String matricula, Coche coche){
12         if (cochesEnTaller.containsKey(matricula)) {
13             System.out.println(x:"Coche con esa matricula ya existe");
14             return false;
15         }else{
16             cochesEnTaller.put(matricula, coche);
17             System.out.println(x:"Coche registrado correcto");
18             return true;
19         }
20     }
21 }
22
```

The 'anadeElemento' method is highlighted in the code.

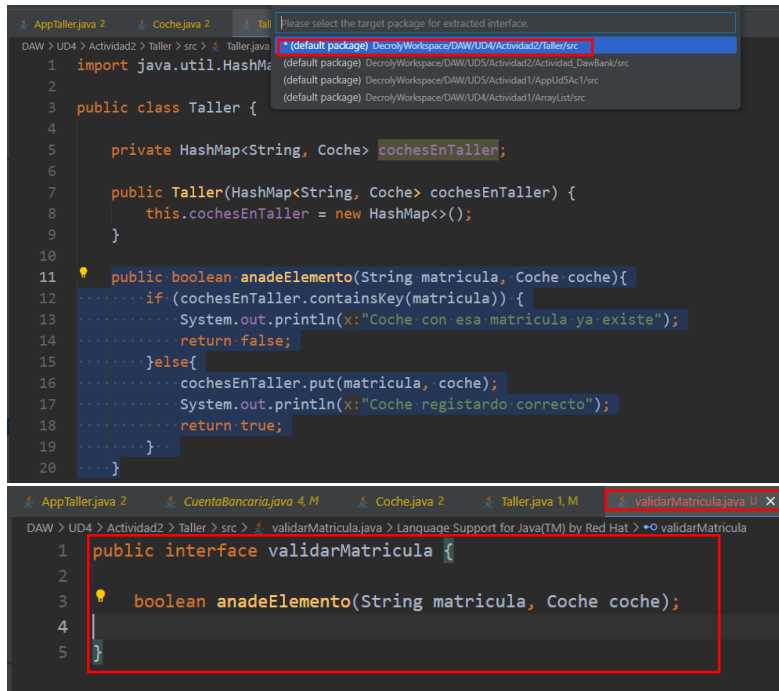
Después de elegir extract, seleccionamos el método que necesitamos.



```
1 import java.util.HashMap;
2
3 public class Taller {
4
5     private HashMap<String, Coche> cochesEnTaller;
6
7     public Taller(HashMap<String, Coche> cochesEnTaller) {
8         this.cochesEnTaller = new HashMap<>();
9     }
10
11     public boolean anadeElemento(String matricula, Coche coche){
12         if (cochesEnTaller.containsKey(matricula)) {
13             System.out.println(x:"Coche con esa matricula ya existe");
14             return false;
15         }else{
16             cochesEnTaller.put(matricula, coche);
17             System.out.println(x:"Coche registrado correcto");
18             return true;
19         }
20     }
21 }
22
```

The 'validarMatricula' method name is entered in the refactoring dialog.

Introducimos el nuevo nombre.

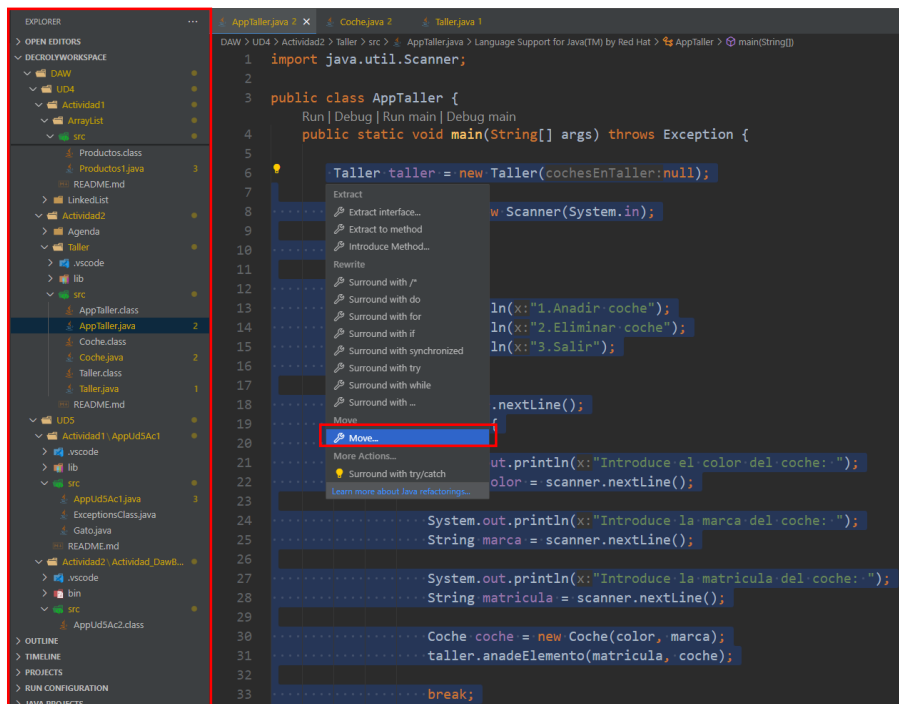


Seleccionamos el proyecto.

Y nos crea una interface validarMatricula con método para añadirElemento.

Mover Archivos o Clases

Puedes mover una clase o archivo de un paquete o directorio a otro para organizar mejor tu proyecto.



Es posible mover los clases, códigos, o archivos por refactor "Move" o también por el "Explorer" arrastrar y dejar en el sitio que nos interesa.

Eliminar Código Muerto (Safety Delete)

Elimina elementos que no se usan en el código.

La manera mas efectiva puede ser instalar algún plugin.

O también siguientes posibilidades:

- **Eliminar Variables No Utilizadas:** Si una variable no es utilizada en ninguna parte del código, elimínala.
- **Puedes buscar referencias a la variable con la función de búsqueda global de VS Code (Ctrl+Shift+F)** para verificar que no se usa en ningún otro lugar.
- **Eliminar Métodos No Usados:** Si tienes métodos que no se invocan en ninguna parte de tu código, también es recomendable eliminarlos. Usa la función de renombrar símbolo (F2) para cambiar el nombre de un método y verificar si alguna parte del código lo usa.
- **Eliminar Importaciones No Utilizadas:** Las importaciones que no se utilizan en el archivo pueden ser eliminadas. Esto puede hacerse manualmente, o también puedes utilizar la funcionalidad integrada de VS Code para limpiar las importaciones.
- **Comando para eliminar importaciones no utilizadas:** Si usas Java con el soporte adecuado de extensiones, puedes hacer clic derecho en tu archivo .java y seleccionar "Organize Imports", lo cual eliminará cualquier importación innecesaria.

Refactorización Manual de Métodos y Parámetros

- **Reducir el Número de Parámetros**

Si un método tiene demasiados parámetros, puede ser una señal de que se debe agrupar algunos de ellos. Una forma común de hacerlo es utilizando un objeto para encapsular varios parámetros relacionados.

- **Eliminar Parámetros No Utilizados**

Asegúrate de que todos los parámetros de un método sean necesarios. Si un parámetro no es utilizado dentro del método, elimínalo.

- **Usar Parámetros Opcionales**

Si tienes muchos parámetros que se pasan con valores predeterminados o que pueden ser opcionales, puedes agruparlos en una clase o utilizar patrones como Builder para mejorar la legibilidad.