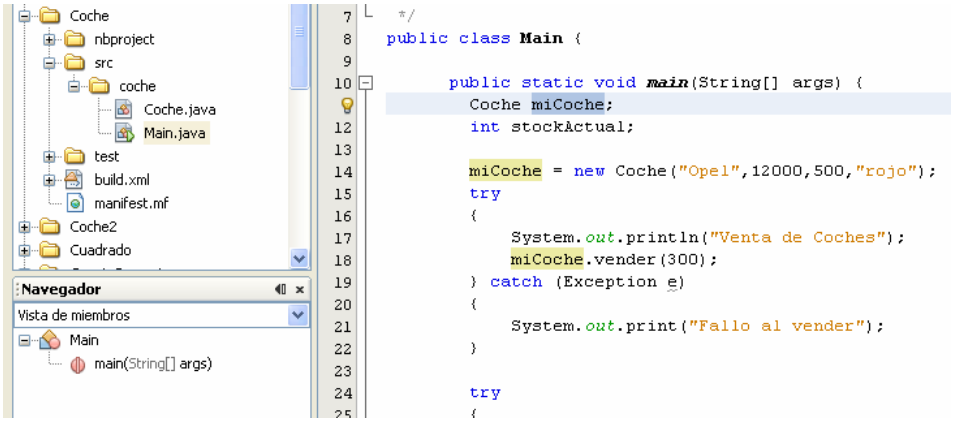
En este documento vamos a ver distintos ejemplos de REFACTORIZACIÓN, ANÁLISIS DE CÓDIGO Y DOCUMENTACIÓN de un proyecto JAVA. Estos ejemplos se van a realizar para el proyecto Coche.

Lo primero que debemos de hacer es Abrir el proyecto Coche y seleccionar el fichero Main.java.

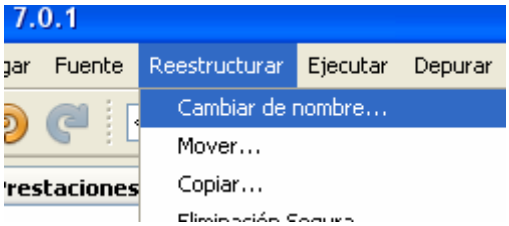
REFACTORIZACIÓN.

1. **Cambio de nombre a una variable.**

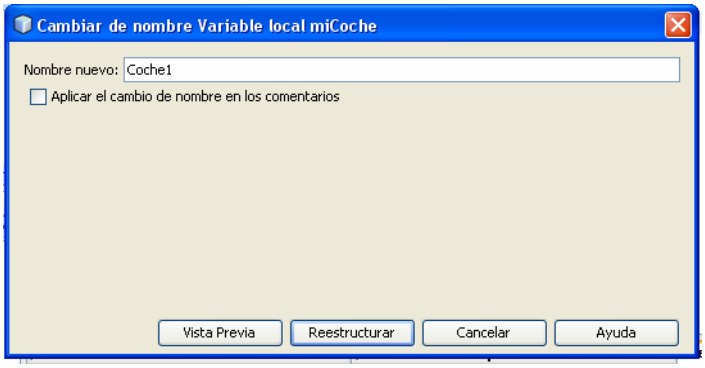
Seleccionamos con el ratón la variable que queremos cambiar de nombre: (En este caso la variable es miCoche.)



A continuación seleccionamos la opción: **Reestructurar -> cambiar de nombre**.



En la ventana que nos aparece ponemos el nombre que queremos cambiar, en este caso Coche1 y pulsamos el botón Reestructurar.

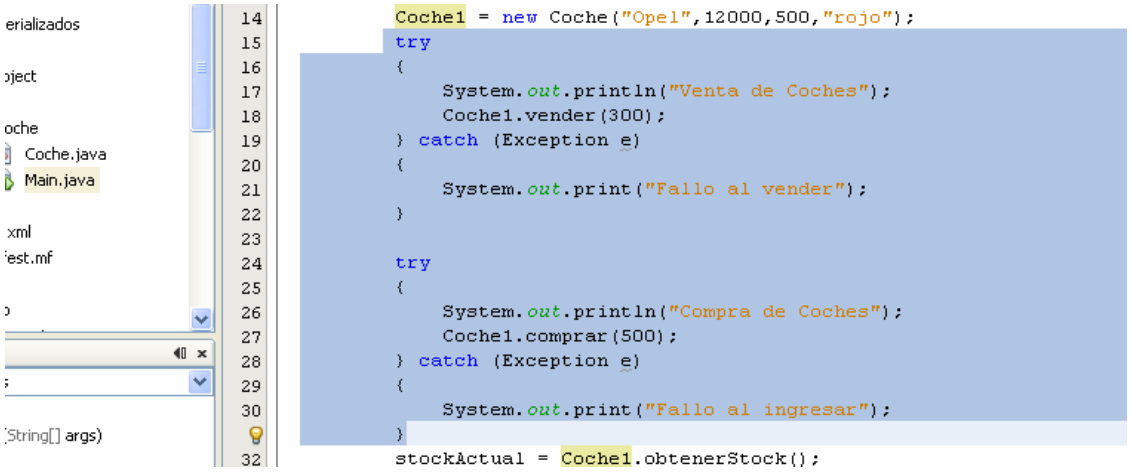


Ahora ya, donde aparecía miCoche, aparece coche1.

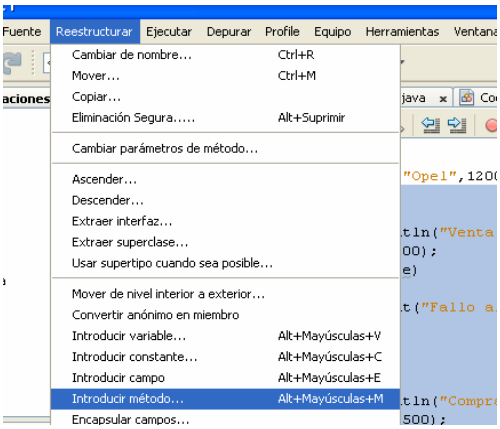
```
public class Main {

    public static void main(String[] args) {
        Coche Coche1;
        int stockActual;

        Coche1 = new Coche("Opel",12000,500,"rojo");
        try
        {
            System.out.println("Venta de Coches");
            Coche1.vender(300);
        } catch (Exception e)
        {
            System.out.print("Fallo al vender");
        }
```

2. Introduce el método operativa_coches, que englobe las sentencias de la clase Main que operan con el objeto coche1.

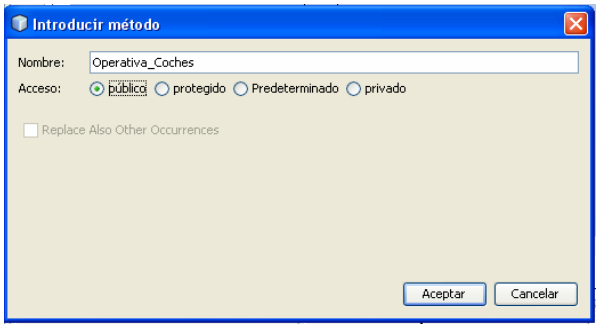
Selecciono las operaciones de compra y venta de coches que se realizan con coche1 y que quiero englobarlas en un método.



Seleccionamos Reestructurar->Introducir método.



//En la ventana que nos aparece ponemos el nombre que queremos cambiar, en este caso operatoria_coches y pulsamos el botón Reestructurar.



Como podemos observar:

Se ha creado un método con el nombre Operativa_Coches(coche1) y el código seleccionado anteriormente.

```
public static void Operativa_Coches(Coche Coche1) {
    try
    {
        System.out.println("Venta de Coches");
        Coche1.vender(300);
    } catch (Exception e)
    {
        System.out.print("Fallo al vender");
    }

    try
    {
        System.out.println("Compra de Coches");
        Coche1.comprar(500);
    } catch (Exception e)
    {
        System.out.print("Fallo al ingresar");
    }
}
```

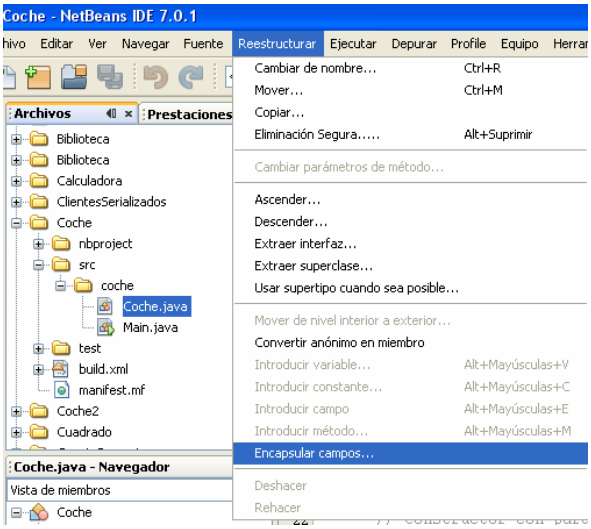
Y en el método Main() se ha sustituido el código que hemos seleccionado por el método Operativa_Coches(coche1);

```
8   public class Main {
9
10      public static void main(String[] args) {
11        Coche Coche1;
12        int stockActual;
13
14        Coche1 = new Coche("Opel",12000,500,"rojo");
15        Operativa_Coches(Coche1);
16        stockActual = Coche1.obtenerStock();
17        System.out.println("El stock actual es"+ stockActual );
18    }
```

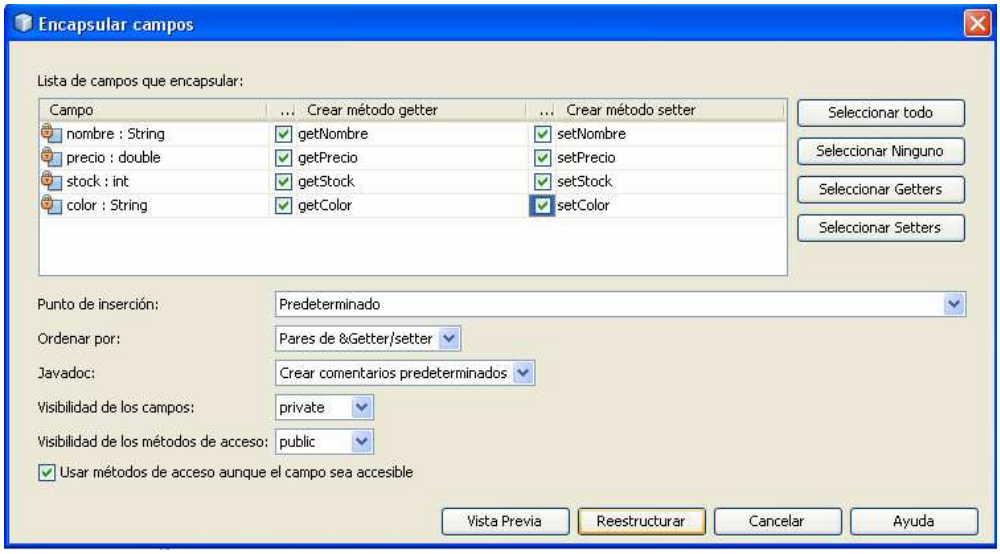
3. Encapsulación de los campos de la clase Coche.

Pinchamos sobre el fichero Coche.java.

Seleccionamos Reestructurar-> Encapsular campos.



En la ventana que nos aparece seleccionamos:



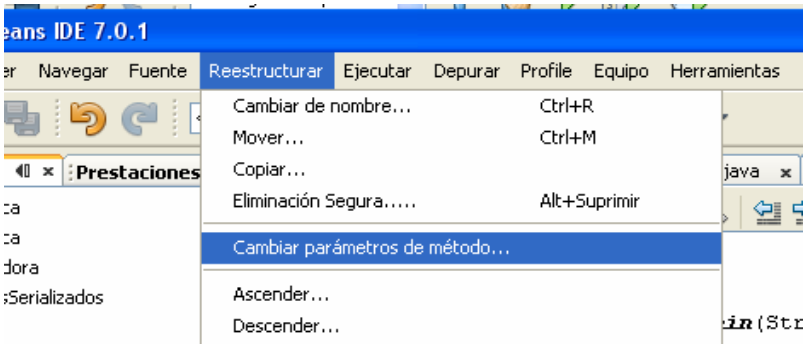
Y pulsamos el botón Reestructurar.

Podemos observar en el fichero que se han creado todos los métodos seleccionados.

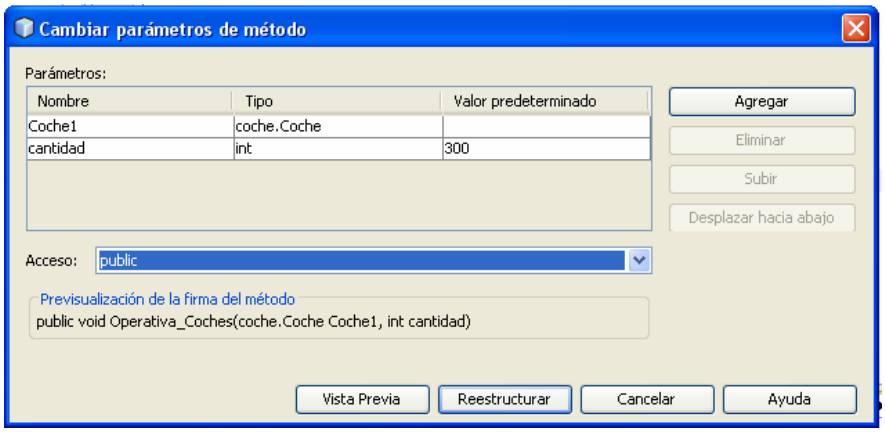
```
69          * @return the nombre
70          */
71         public String getNombre() {
72             return nombre;
73         }
74
75         /**
76          * @param nombre the nombre to set
77          */
78         public void setNombre(String nombre) {
79             this.nombre = nombre;
80         }
81
82         /**
83          * @return the precio
84          */
85         public double getPrecio() {
86             return precio;
87         }
```

4. Añadir parámetros al método Operativa_Coches.

Seleccionamos la opción Reestructurar->Cambiar parámetros de método, del menú de NetBeans, y procedemos a añadir un nuevo parámetro.



Nos aparece la siguiente pantalla donde añadimos el parámetro cantidad de tipo int y con valor predeterminado 300.



Como podemos observar en el método Operativa_Coches aparece el nuevo parámetro cantidad.

```
public static void Operativa_Coches(Coche Coche1, int cantidad) {
    try
    {
        System.out.println("Venta de Coches");
        Coche1.vender(300);
    } catch (Exception e)
    {
        System.out.print("Fallo al vender");
    }
```

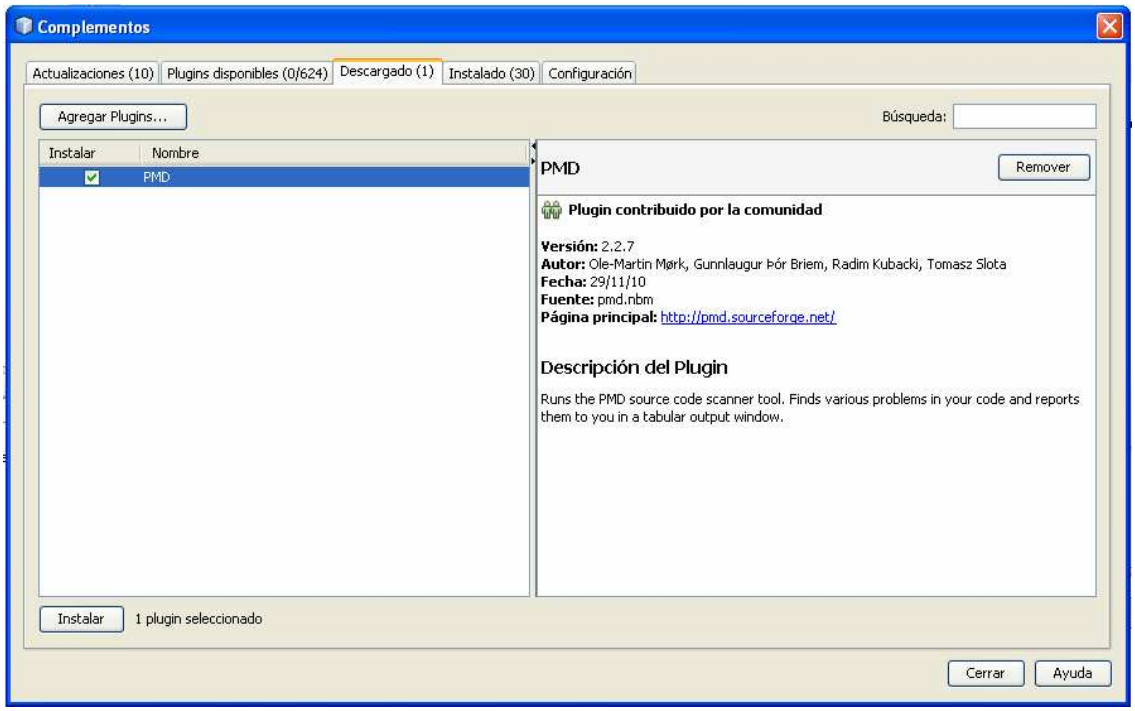
Ahora puedo sustituir el valor 300 de Coche1.vender(300) por cantidad quedando:

```
public static void Operativa_Coches(Coche Coche1, int cantidad) {
    try
    {
        System.out.println("Venta de Coches");
        Coche1.vender(cantidad);
    } catch (Exception e)
    {
        System.out.print("Fallo al vender");
    }
```

ANÁLISIS DE CÓDIGO.

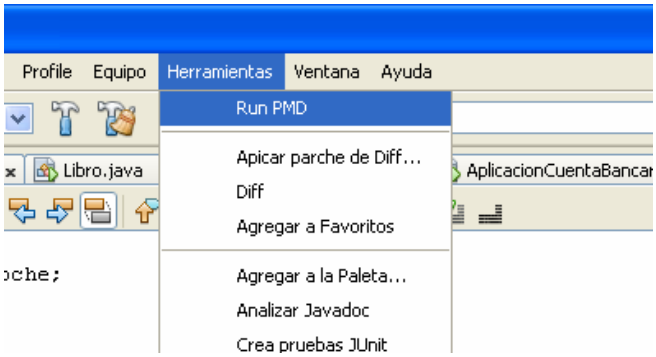
5. Instalación del analizador de código PMD.

Dado que este analizador no está instalado en Netbeans por defecto. Lo primero que haremos será descargar el plugin desde: http://sourceforge.net/projects/pmd/files/ y procederemos a su instalación.

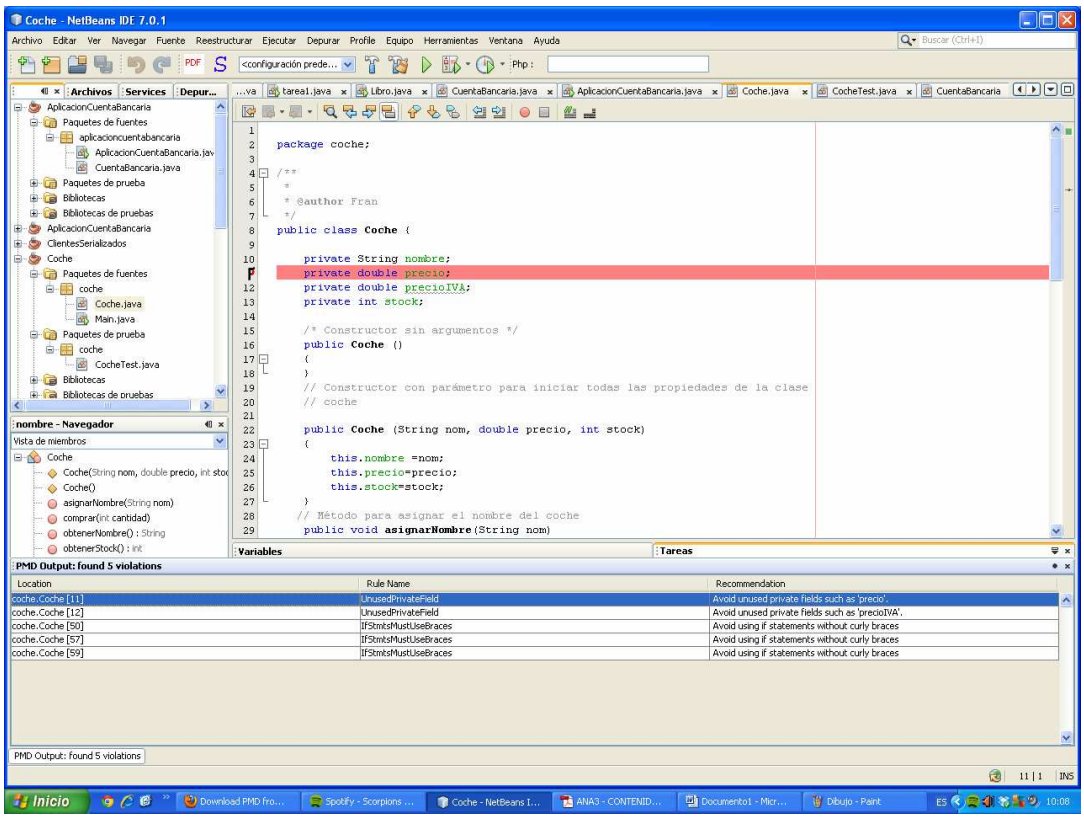


6. Uso del analizador de código PMD.

Una vez que lo hemos instalado, si queremos analizar nuestro código con PMD, y obtener el informe del análisis, pulsamos el botón derecho del ratón sobre el directorio que contiene los ficheros de código, en la vista de proyectos y elegimos: Herramientas - Ejecutar PMD.



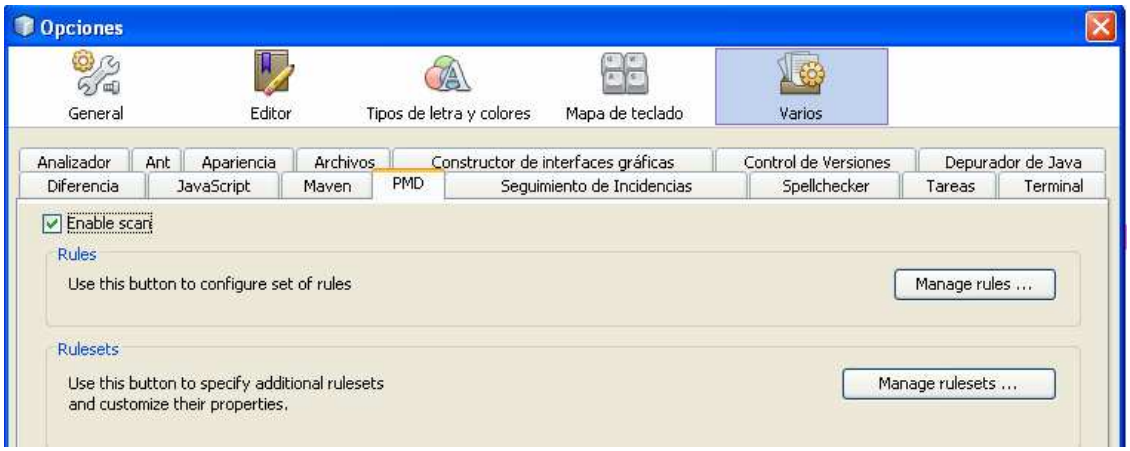
Mientras se analiza el código, el plug-in mostrará una barra de progreso en la esquina inferior derecha. El informe producido contiene la localización, el nombre de la regla que no se cumple y la recomendación de cómo se resuelve el problema. El informe PMD, nos permite navegar por el fichero de clases y en la línea donde se ha detectado el problema.En el número de línea, veremos una marca PMD. Si se posiciona el ratón encima de ella, veremos un tooltip con la descripción del error.



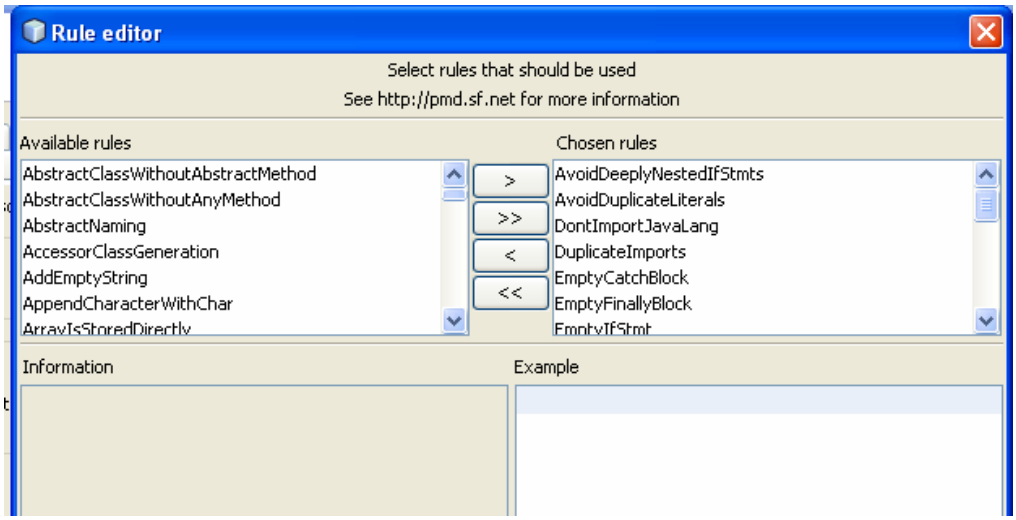
El análisis lo debemos de realizar tanto para el fichero main.java como para el fichero Coche. Java.

7. Configuración del analizador de código PMD para que se permita el escaneado automático del proyecto, a intervalos regulares.

Para configurar el PDM accedemos **Herramientas - Opciones - Varios – PMD** y activamos la casilla **Enable scan**.



8. Si queremos añadir o quitar reglas lo haríamos en la ventana que aparece al pulsar Manage rules.



DOCUMENTACIÓN

9. Inserción de comentarios Javadoc en el proyecto.

Los comentarios de una clase deben comenzar con /** y terminar con */ . Entre la información que debe incluir un comentario de clase debe incluirse, al menos las etiquetas @autor y @version, donde @autor identifica el nombre del autor o autora de la clase y @version, la identificación de la versión y fecha.

```java
package coche;

/**
 *
 * @author Fran
 * @version 1.0 20/2/2012
 */
public class Coche {

    private String nombre;
    private double precio;
```

Dentro de la clase, también se documentan **los constructores y los métodos**. Al menos se indican las etiquetas:
– **@param**: seguido del nombre, se usa para indicar cada uno de los parámetros que tienen el constructor
o método.
– **@return**: si el método no es void, se indica lo que devuelve.
– **@exception**: se indica el nombre de la excepción, especificando cuales pueden lanzarse.
- @throws: se indica el nombre de la excepción, especificando las excepciones que pueden lanzarse.

Por ejemplo el método vender se podría documentar con etiquetas javadoc de la siguiente forma:

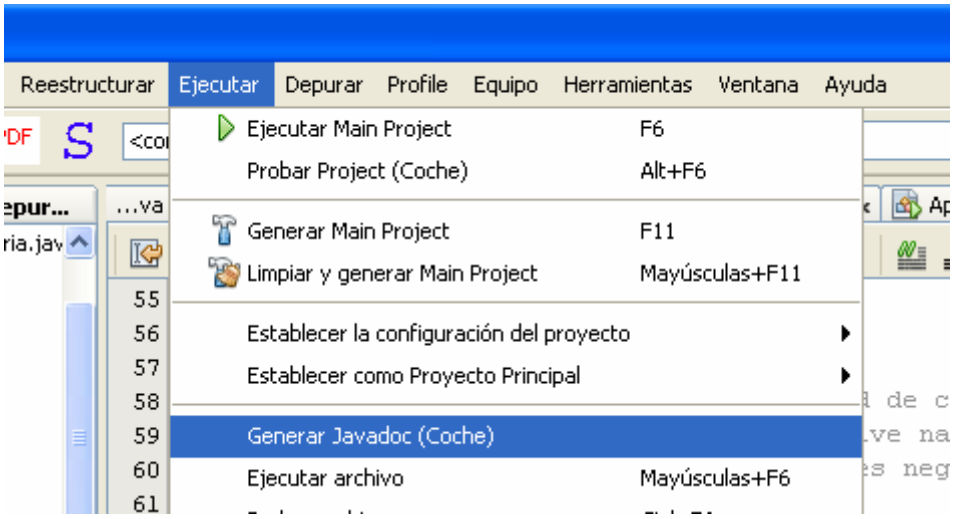
```java
/**
 * @param cantidad. Indica la cantidad de coches que queremos vender.
 * @return void. Este método no devuelve nada.
 * @throws Exception. Si la cantidad es negativa mayor que el stock se produce una
 * excepción de tipo Exception

*/

public void vender (int cantidad) throws Exception
{
    if (cantidad <= 0)
        throw new Exception ("No se puede vender una cantidad negativa de coches");
    if (obtenerStock()< cantidad)
        throw new Exception ("No se hay suficientes coches para vender");
    stock = stock - cantidad;
}
```

Esto habría que hacerlo para cada método y constructor de la clase Coche (Coche.java) y también habría que documentar el programa principal main (fichero main.java)

10. Generación del documento Javadoc para todo el proyecto.

Una vez documentado el código con las etiquetas javadoc para generar páginas HTML de documentación a partir de los comentarios incluidos en el código fuente ejecutamos:

Ejecutar->Generar Javadoc



Y por ejemplo el método vender que hemos documentado aparecería documentado de la siguiente forma:

vender

```
public void vender(int cantidad)
            throws java.lang.Exception
```

Parameters:

`cantidad` - Indica la cantidad de coches que queremos vender.

Throws:

`java.lang.Exception` - Si la cantidad es negativa mayor que el stock se produce una excepción de tipo Exception