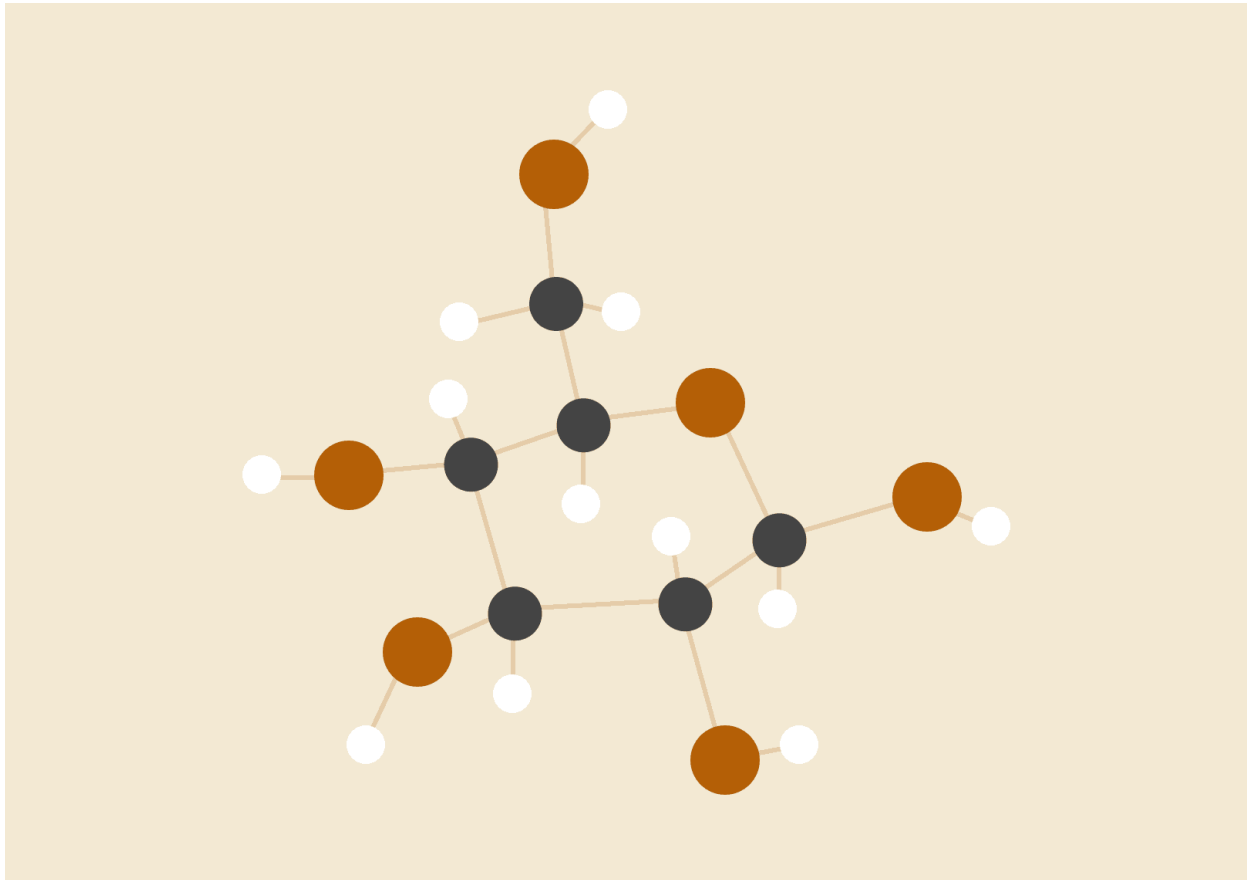


# Optimización Y Documentación

*ED04 Tarea E1*



**Juan Antonio García Luna**

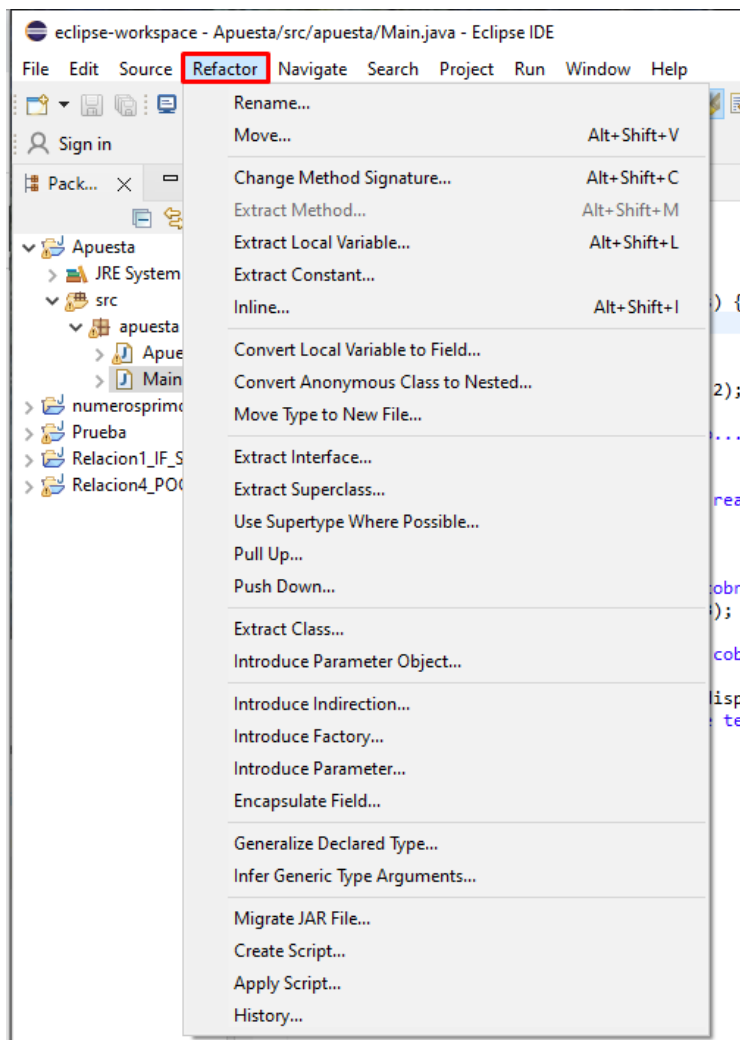
1ºDAM

# ÍNDICE

<b>Refactorización</b>	<b>2</b>
Ejercicio 1	2
Ejercicio 2	4
Ejercicio 3	6
Ejercicio 4	9
<b>Analizador de código</b>	<b>12</b>
Ejercicio 5	12
Ejercicio 6	15
Ejercicio 7	16
Ejercicio 8	17
<b>JavaDoc</b>	<b>21</b>
Ejercicio 9	21
Ejercicio 10	25

## Refactorización

Para las opciones de refactorización en Eclipse hay un submenú tanto en la parte superior como haciendo clic derecho sobre cualquier parte del código, este menú es el que usaremos en los ejercicios de refactorización.

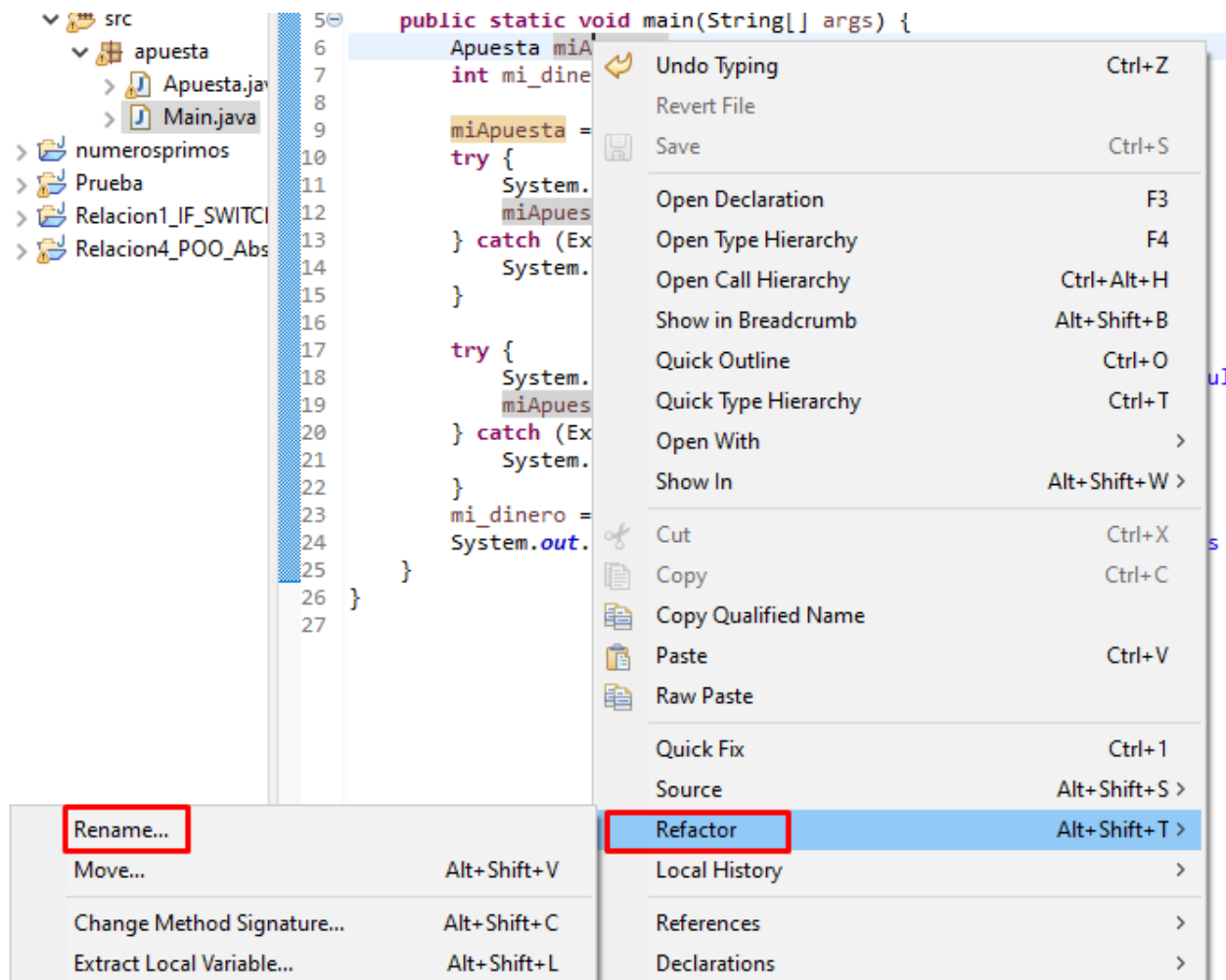


### Ejercicio 1

Cambia el nombre de la variable “miApuesta” por "laApuesta".

Para poder cambiar el nombre de una variable que usamos en todo el código reiteradas

veces podemos usar la opción rename del menú mencionado anteriormente.



Veremos que se señala la variable todas las veces que sale y si cambiamos el nombre se cambiará a la vez en todas las iteraciones de la variable.

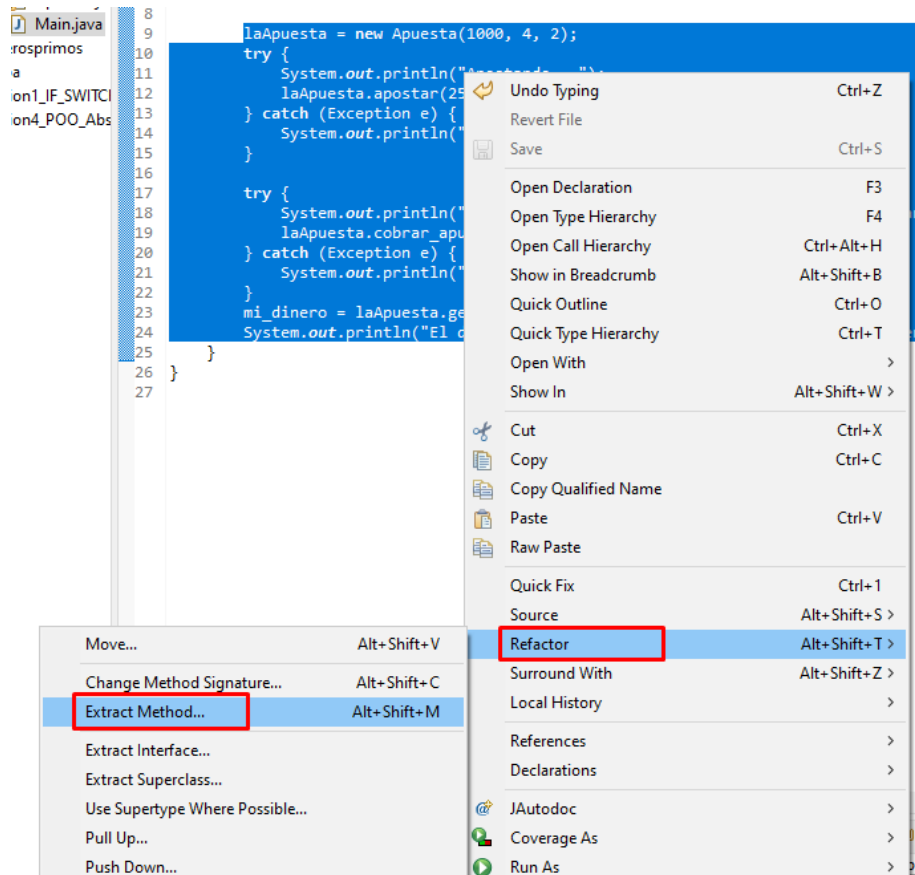
```
public static void main(String[] args) {
    Apuesta laApuesta;
    int mi_dinero;
    laApuesta = new Apuesta(1000, 4, 2);
    try {
        System.out.println("Apostando...");
        laApuesta.apostar(25);
    } catch (Exception e) {
        System.out.println("Fallo al realizar la Apuesta");
    }

    try {
        System.out.println("Intento cobrar apuesta segun el resultado del partido");
        laApuesta.cobrar_apuesta(2, 3);
    } catch (Exception e) {
        System.out.println("Fallo al cobrar la apuesta");
    }
    mi_dinero = laApuesta.getDinero_disp();
    System.out.println("El dinero que tengo tras las apuestas es " + mi_dinero);
}
```

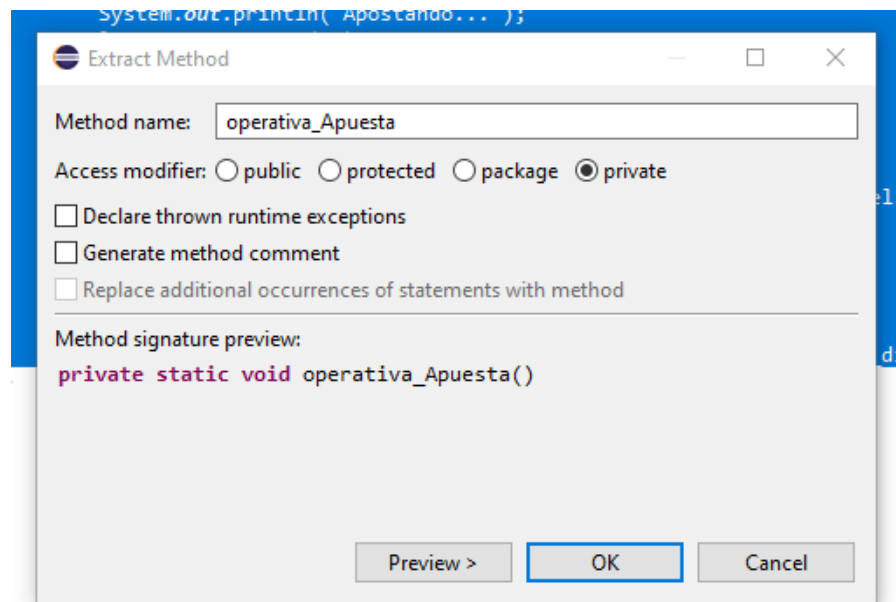
## Ejercicio 2

Introduce el método operativa `Apuesta`, que englobe las sentencias de la clase `Main` que operan con el objeto `laApuesta`.

Para este método de refactorización usaremos también una opción del menú mencionado anteriormente. En esta ocasión seleccionamos la parte del código que queremos exportar a un método y usaremos la opción `Extract Method...`



Al clicar esta opción nos aparecerá una ventana que nos pedirá que elegimos algunas propiedades del método que se va a crear como el nombre o el tipo del que será.



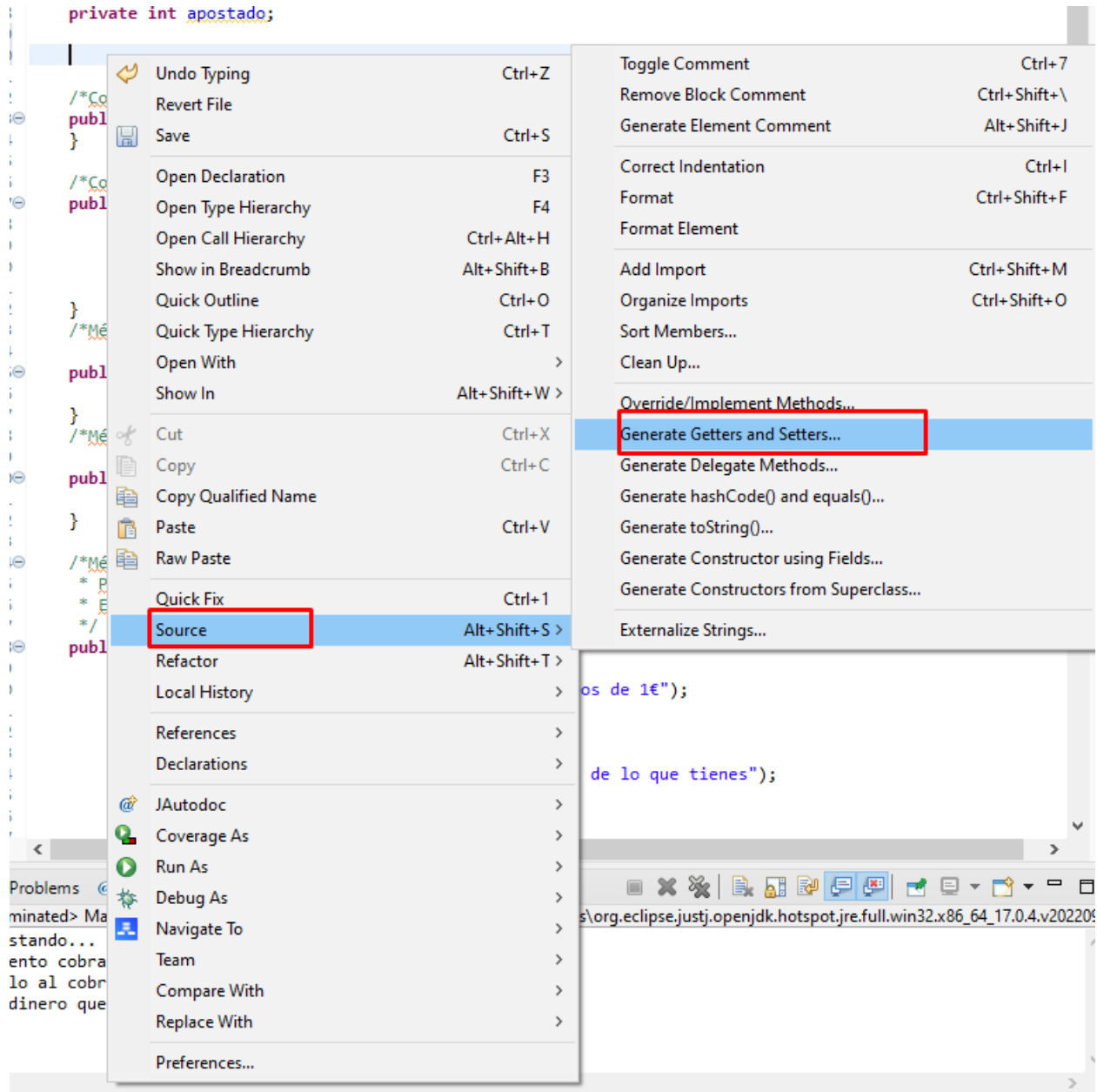
Al pulsar en OK lo que hará será crear el método bajo el método Main, añadirá todas las líneas de código que teníamos seleccionadas a ese nuevo método y creará una llamada al nuevo método en el lugar en el que estaba el código.

```
public class Main {  
    public static void main(String[] args) {  
        Apuesta laApuesta;  
        int mi_dinero;  
        operativa_Apuesta();  
    }  
    private static void operativa_Apuesta() {  
        Apuesta laApuesta;  
        int mi_dinero;  
        laApuesta = new Apuesta(1000, 4, 2);  
        try {  
            System.out.println("Apostando...");  
            laApuesta.apostar(25);  
        } catch (Exception e) {  
            System.out.println("Fallo al realizar la Apuesta");  
        }  
  
        try {  
            System.out.println("Intento cobrar apuesta segun el resultado del partido");  
            laApuesta.cobrar_apuesta(2, 3);  
        } catch (Exception e) {  
            System.out.println("Fallo al cobrar la apuesta");  
        }  
        mi_dinero = laApuesta.getDinero_disp();  
        System.out.println("El dinero que tengo tras las apuestas es " + mi_dinero);  
    }  
}
```

## Ejercicio 3

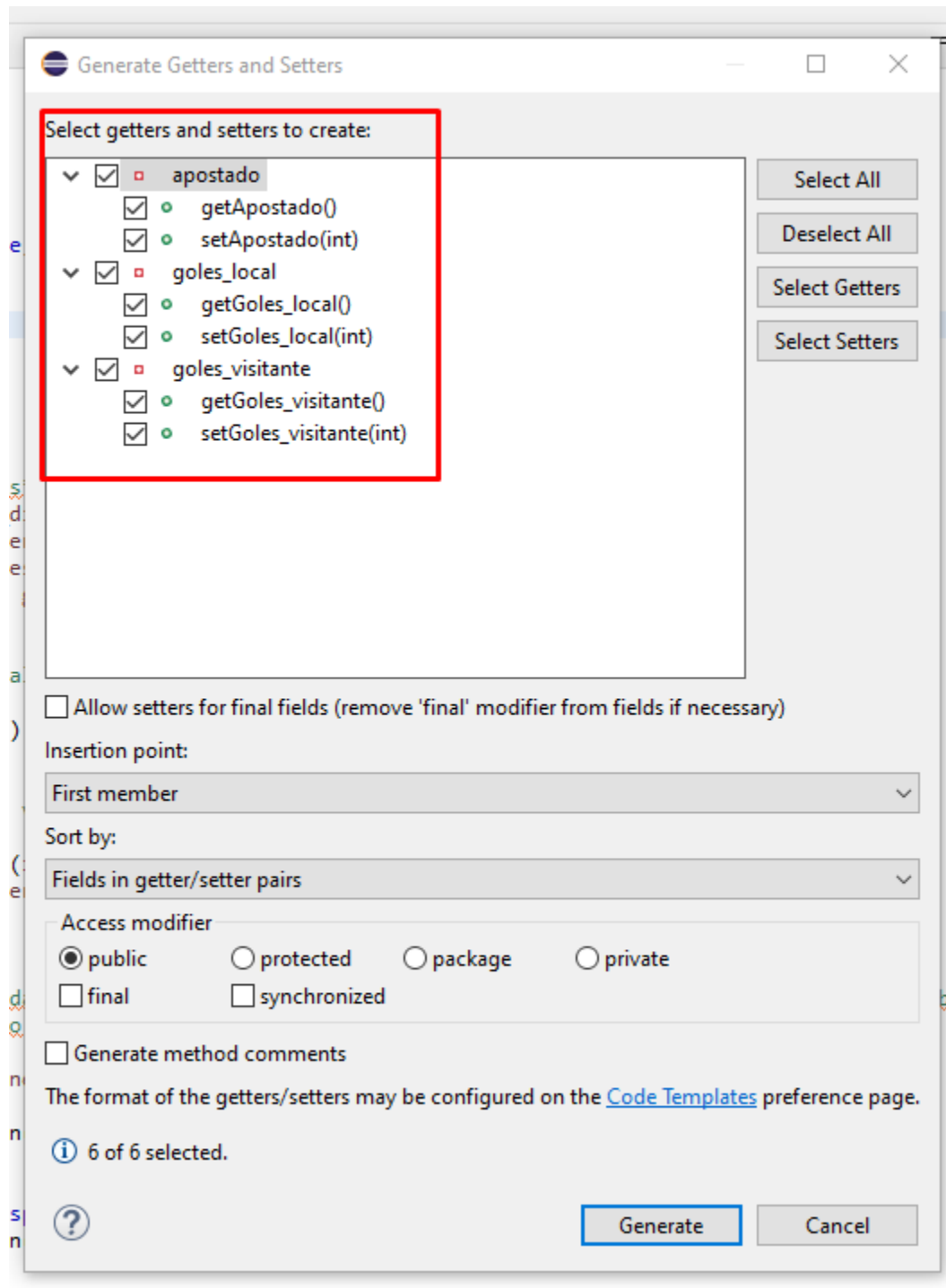
**Encapsula todos los atributos de la clase Apuesta.**

Para encapsular los atributos de la clase Apuesta crearemos unos métodos para ver el valor de cada atributo y otro para cambiar el valor de cada atributo, estos métodos se conocen como métodos getter y setter. Para hacerlo haremos clic derecho sobre cualquier parte de la clase y nos iremos al menú Source donde encontraremos una opción que se llama *Generate Getters and Setters*.



Al clicar esta opción se nos abrirá una ventana donde podremos elegir que getters y setters generar.





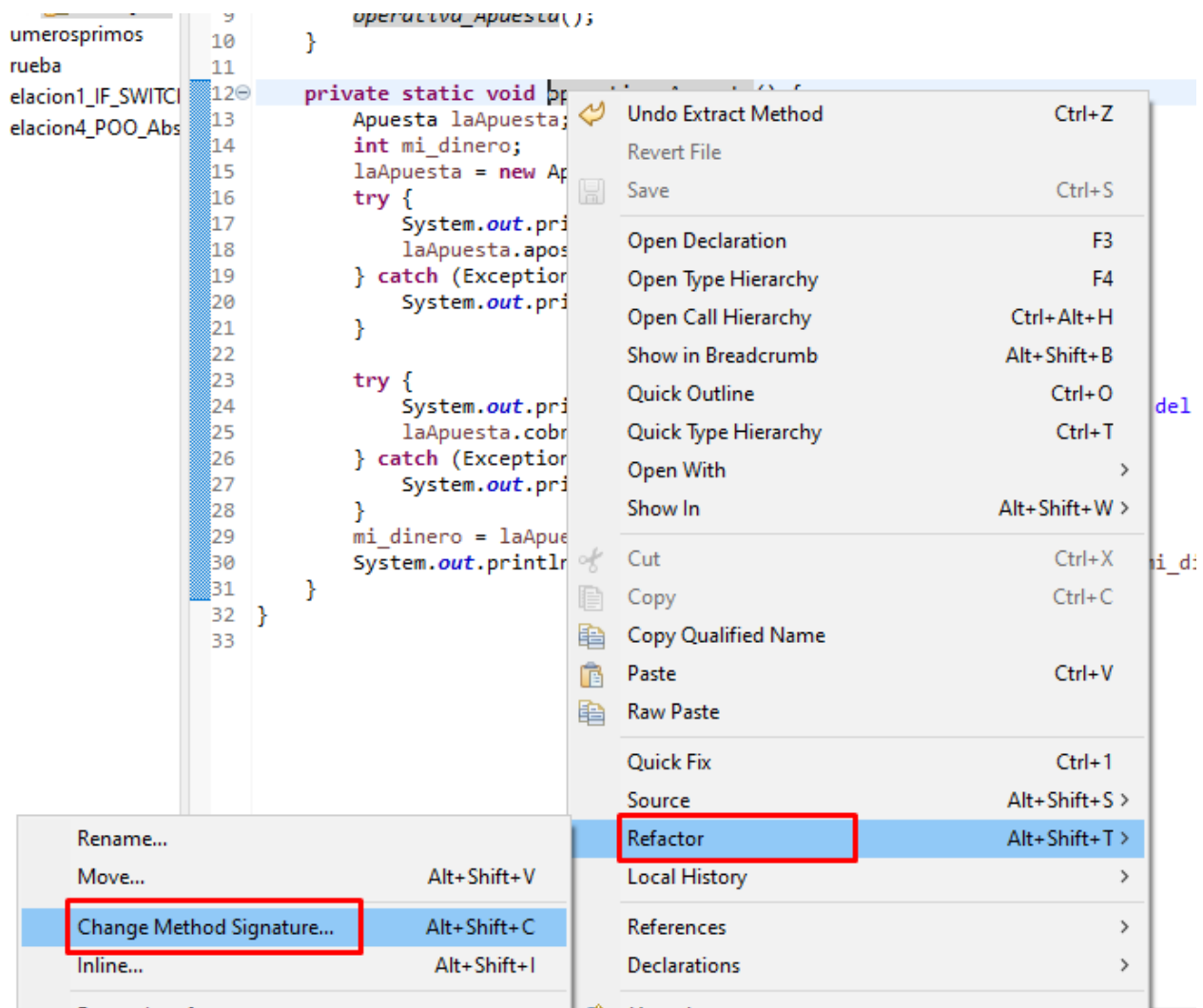
Cuando pulsamos en el botón generar se generarán los métodos de forma automática.

```
public class Apuesta {  
    public int getGoles_local() {  
        return goles_local;  
    }  
    public void setGoles_local(int goles_local) {  
        this.goles_local = goles_local;  
    }  
    public int getGoles_visitante() {  
        return goles_visitante;  
    }  
    public void setGoles_visitante(int goles_visitante) {  
        this.goles_visitante = goles_visitante;  
    }  
    public int getApostado() {  
        return apostado;  
    }  
    public void setApostado(int apostado) {  
        this.apostado = apostado;  
    }  
    private int dinero_disp;  
    private int goles_local;  
    private int goles_visitante;  
    private int apostado;  
}
```

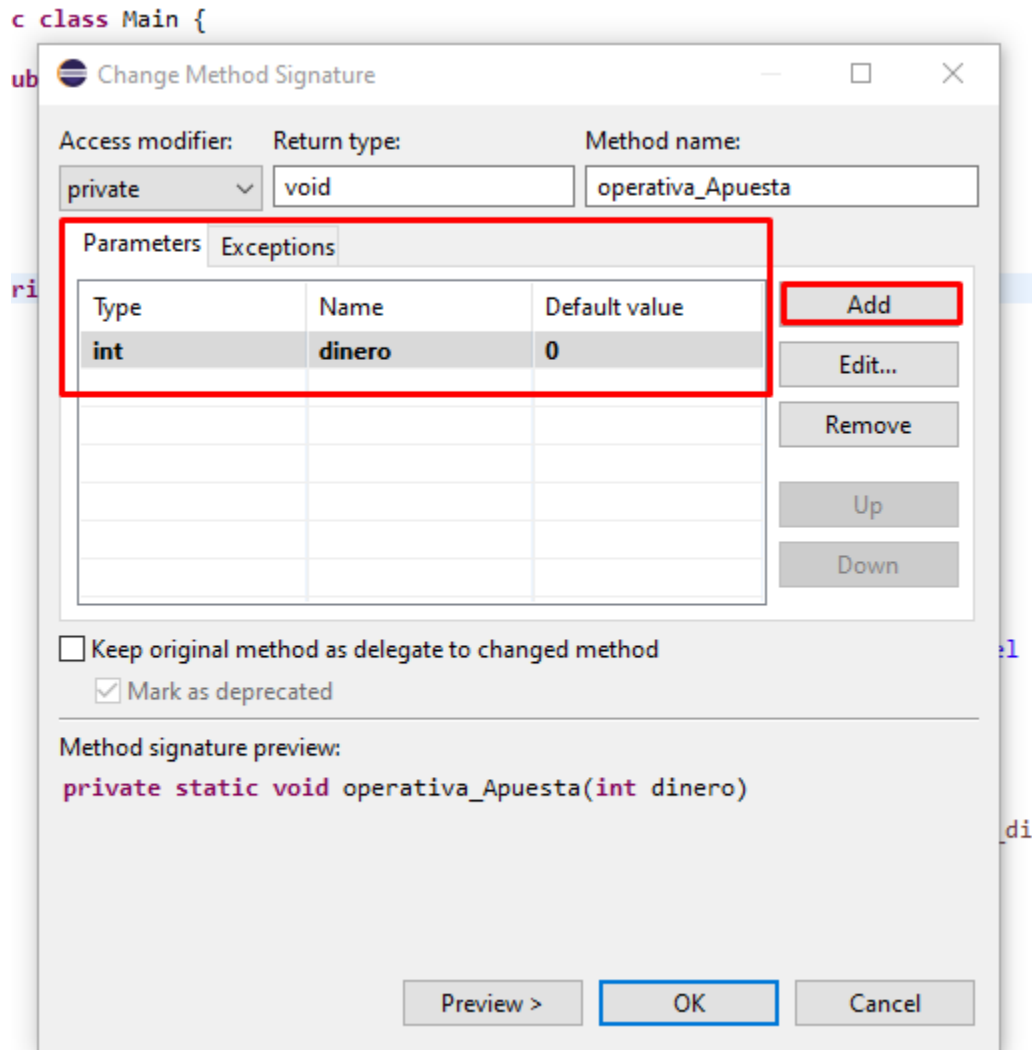
## Ejercicio 4

Añadir un nuevo parámetro al método operativa\_Apuesta, de nombre dinero y de tipo int.

Para añadir un nuevo parámetro a un método siempre podemos escribirlo manualmente pero en este caso usaremos una opción del menú que mencionamos al principio para añadir parámetros. Esta opción se llama *Change Method Signature*.



Al clicar en esta opción se nos abre una ventana que, entre otras opciones, nos permitirá añadir parámetros al método.



Al pulsar OK veremos como se nos ha añadido el parámetro que hemos configurado y en la llamada se ha puesto el valor por defecto que también hemos configurado.

```
    operativa_Apuesta(0);  
}  
  
private static void operativa_Apuesta(int dinero) {  
    Apuesta laApuesta;  
    int mi_dinero;  
    laApuesta = new Apuesta(1000, 4, 2);  
    try {  
        System.out.println("Apostando...");  
        laApuesta.apostar(25);  
    } catch (Exception e) {  
        System.out.println("Fallo al realizar la Apuesta");  
    }  
  
    try {  
        System.out.println("Intento cobrar apuesta segun el resultado del partido");  
        laApuesta.cobrar_apuesta(2, 3);  
    } catch (Exception e) {  
        System.out.println("Fallo al cobrar la apuesta");  
    }  
    mi_dinero = laApuesta.getDinero_disp();  
    System.out.println("El dinero que tengo tras las apuestas es " + mi_dinero);  
}
```

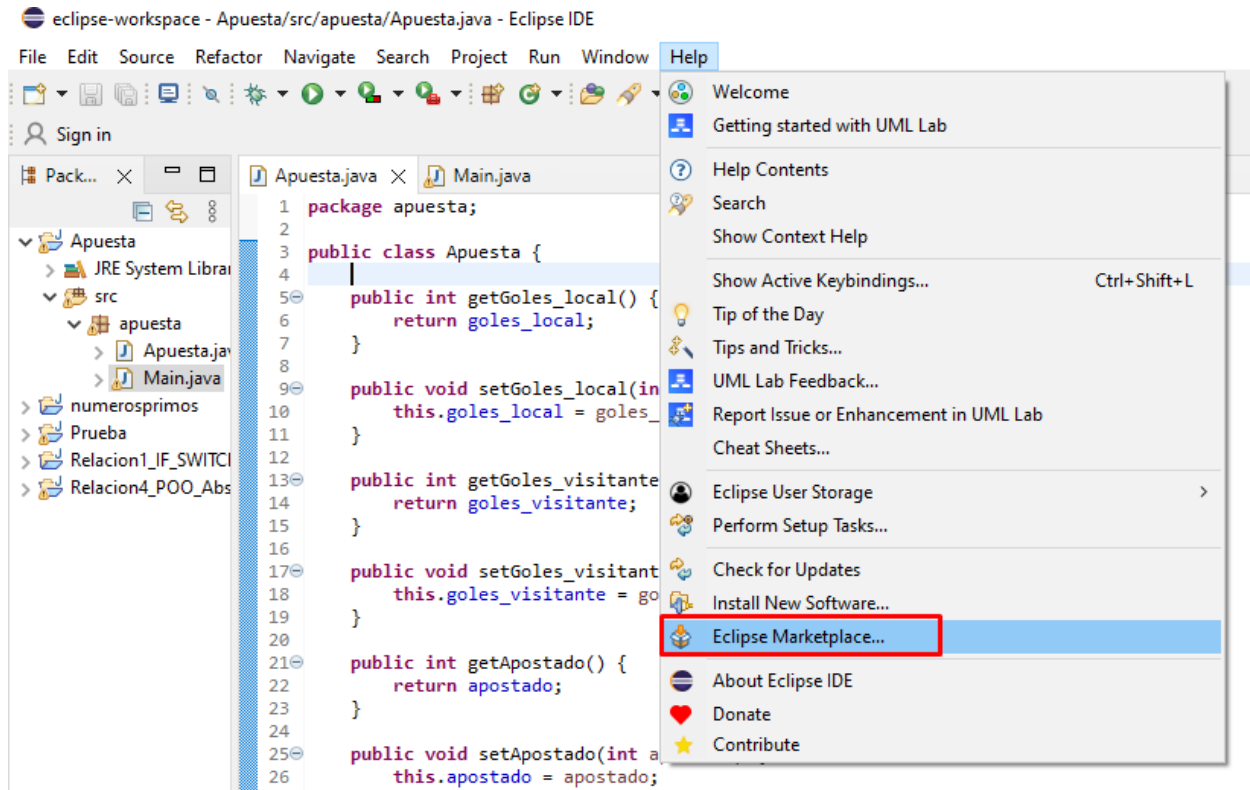
## Analizador de código

Para alzar el código usaremos el plugin PMD que es una herramienta para validar el código según los estándares.

## Ejercicio 5

Descarga e instala el plugin PMD.

Podemos descargar e instalar el plugin desde el marketplace de eclipse.



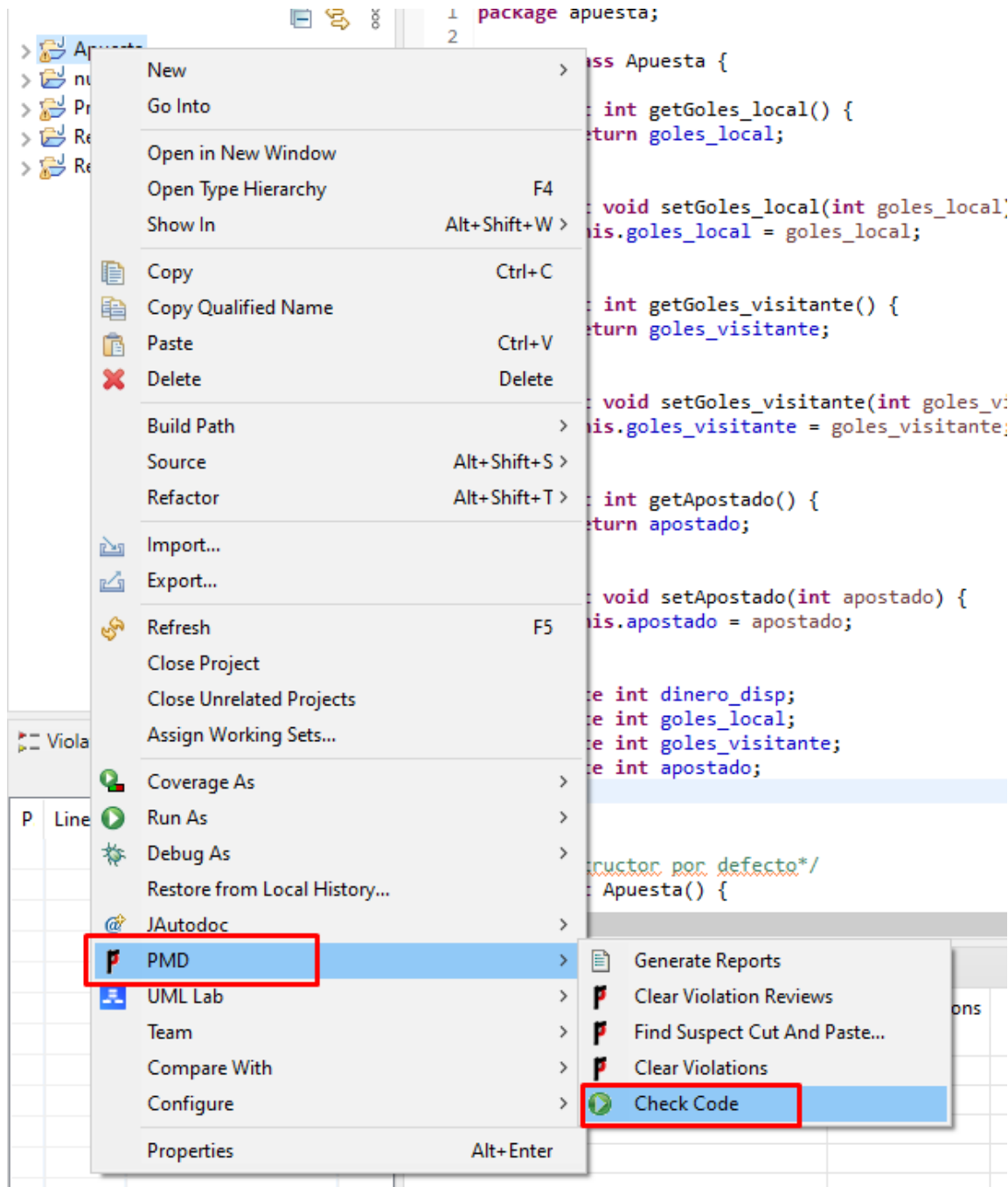
En el marketplace de eclipse podemos buscar PMD y nos saldrán varias opciones, yo elegí la segunda.



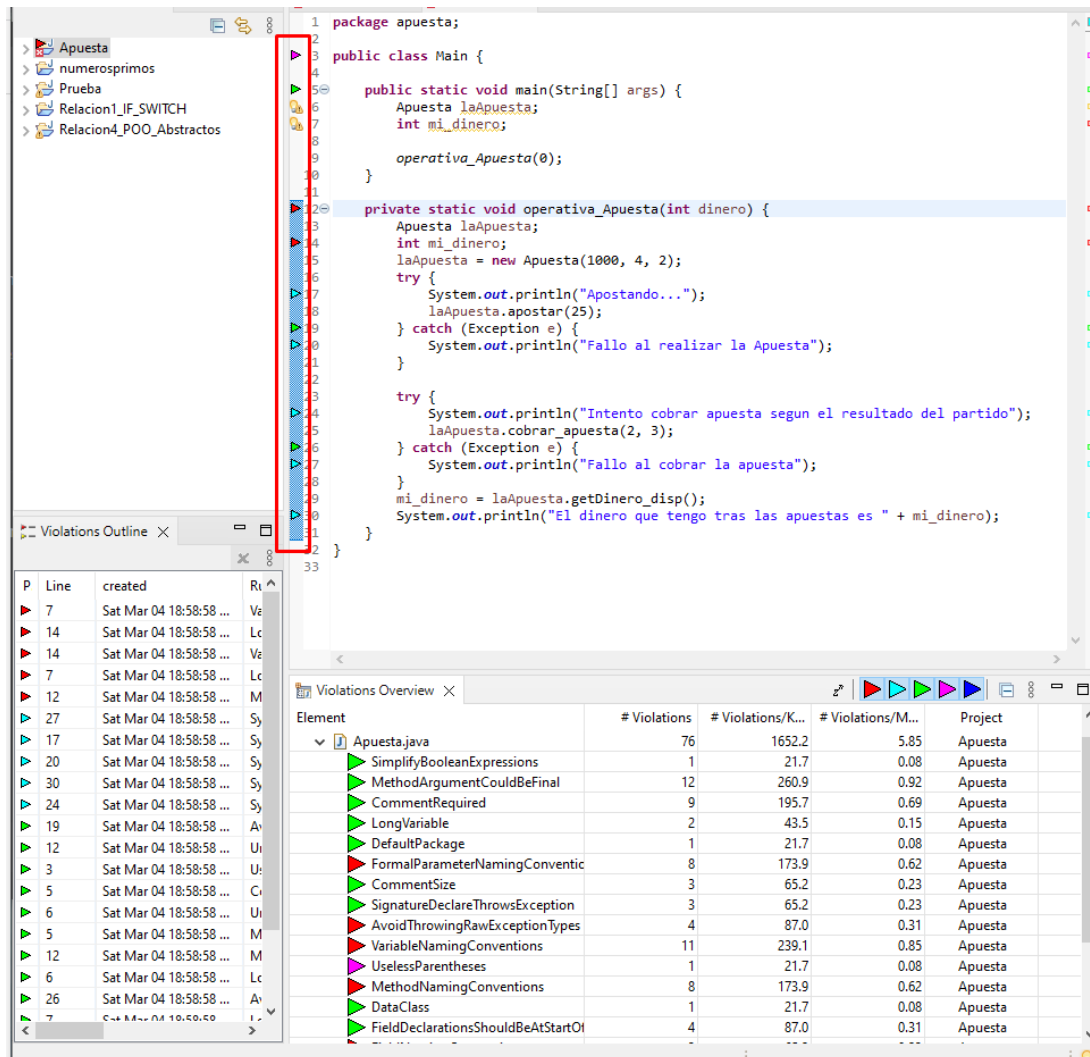
## Ejercicio 6

Ejecuta el analizador de código PMD.

Para ejecutar el PMD tenemos que hacer clic derecho sobre el proyecto que queremos analizar y veremos una opción nueva llamada PMD donde tendrá una subopción para chequear el código.







## Ejercicio 7

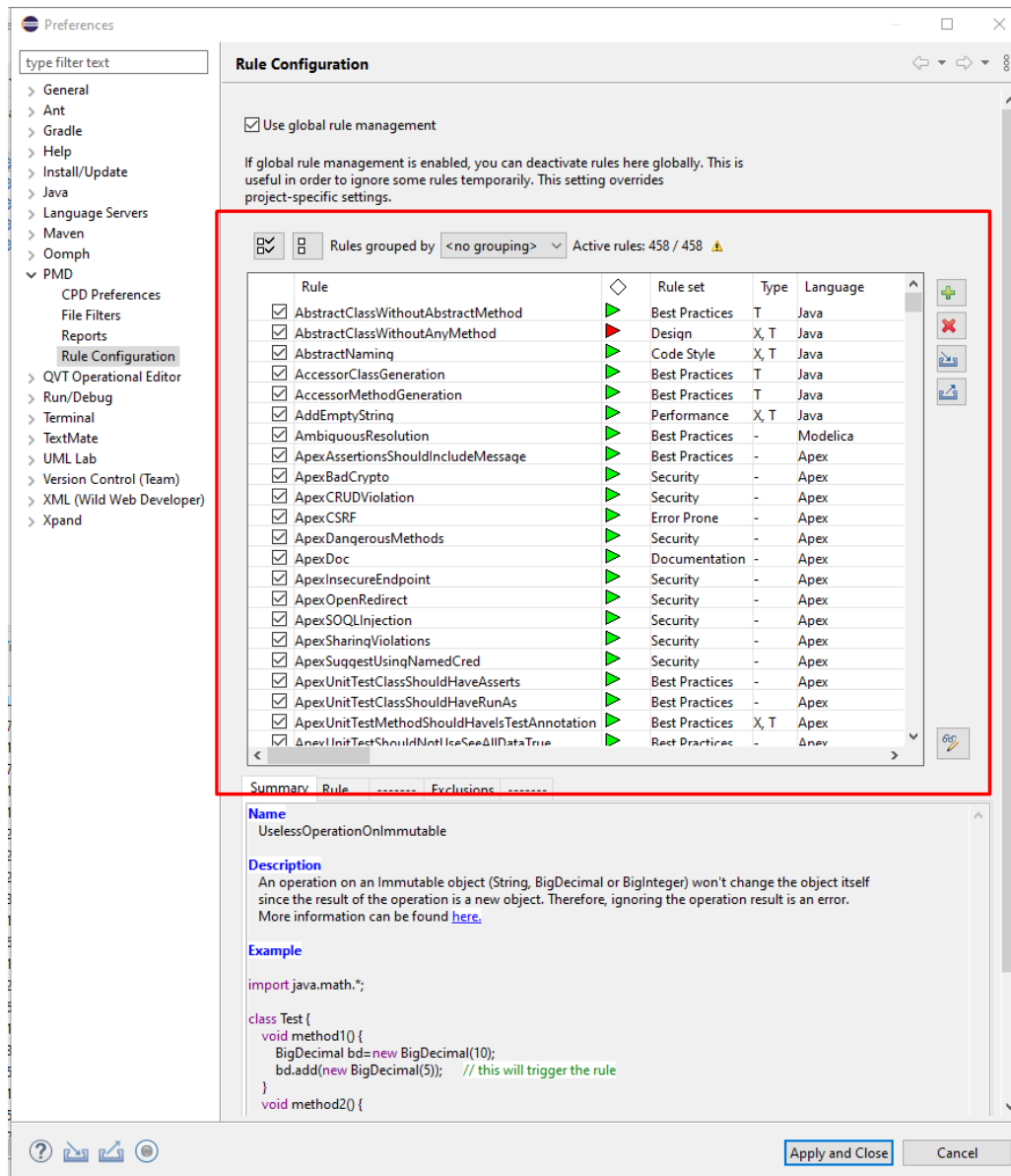
Configura el analizador de código PMD para que se permita el escaneo automático del proyecto, a intervalos regulares.

Investigando no he podido averiguar como hacer que haga escaneos automáticos en intervalos regulares, probablemente el plugin que he instalado no permita esa opción, pero si puedo configurar el plugin para que realice un escaneo cuando guarde algún cambio en el proyecto.

Para configurar el PMD para que haga un escaneo cuando se guarden cambios, tenemos que ir a Window > Preferences y ahí iremos a la parte de PMD, y una de las opciones generales es *Check code after saving*.

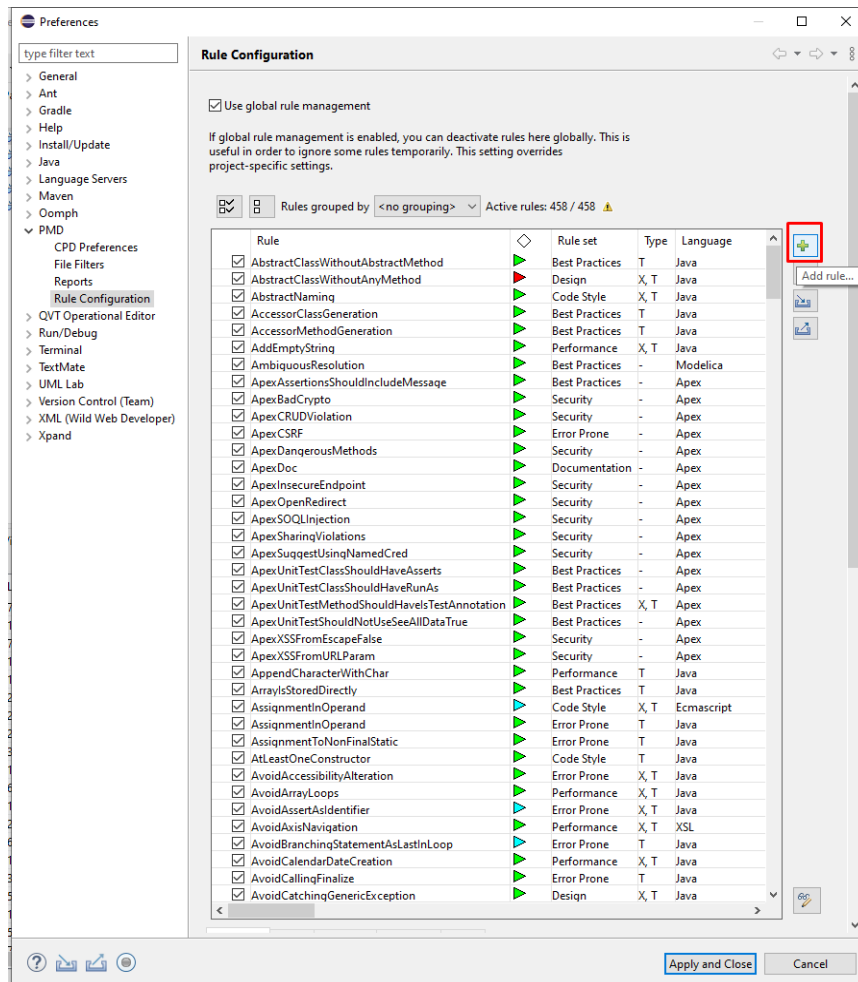


Podemos añadir o quitar reglas del analizador desde Windows > Preferences > PMD > Rule Configuration. Aquí podremos añadir, quitar, importar y exportar reglas, por defecto el analizador viene con un montón de reglas tanto para Java como para otros lenguajes.

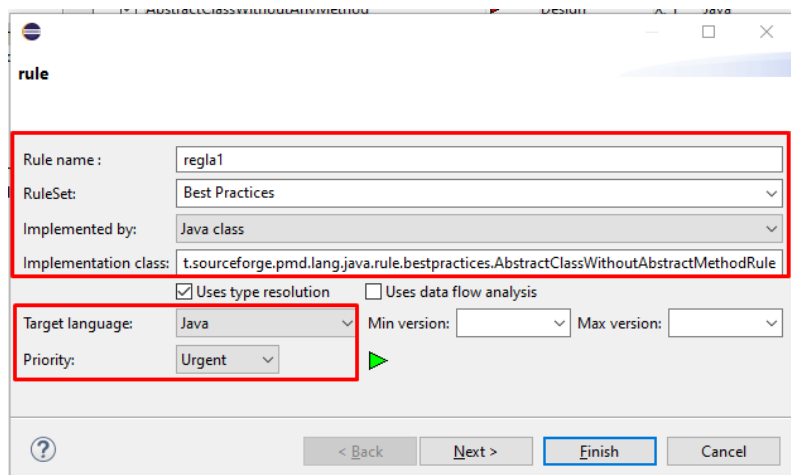


Investigando he comprobado que todas las reglas posibles para Java (por lo menos las más importantes) están ya integradas por defecto en el analizador así que explicaré como se puede crear una regla para el analizador pero no podré crearla yo.

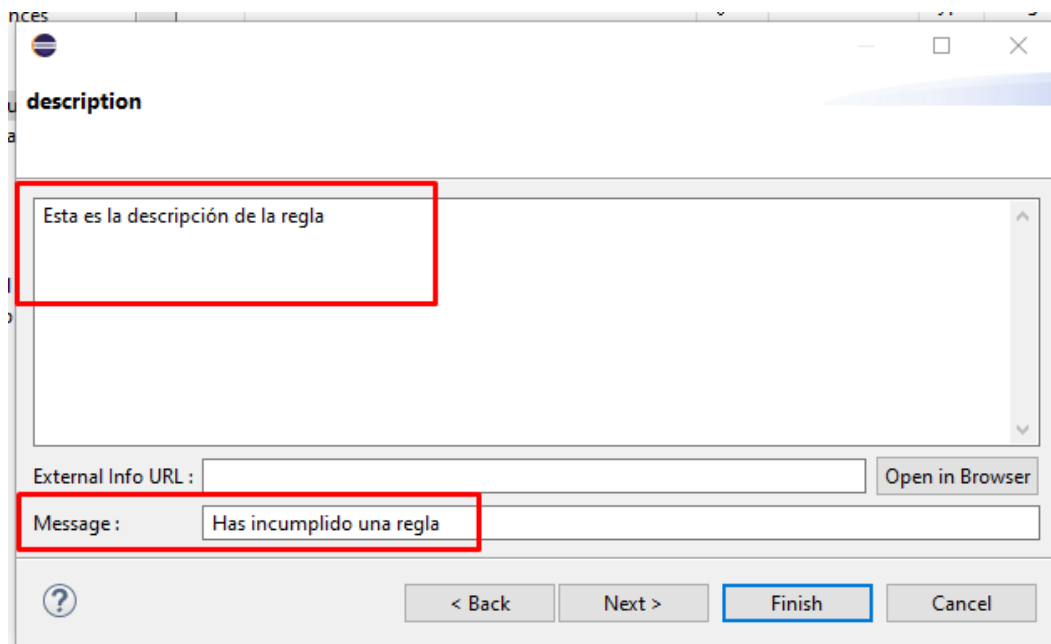
Para crear una regla le daremos al botón de add rule.



Al hacer esto nos aparecerá una ventana donde debemos decidir el nombre de la regla, que tipo de regla será, la implementación de la regla, el lenguaje al que se referirá la regla y la prioridad. Una vez hecho esto pulsamos en next.

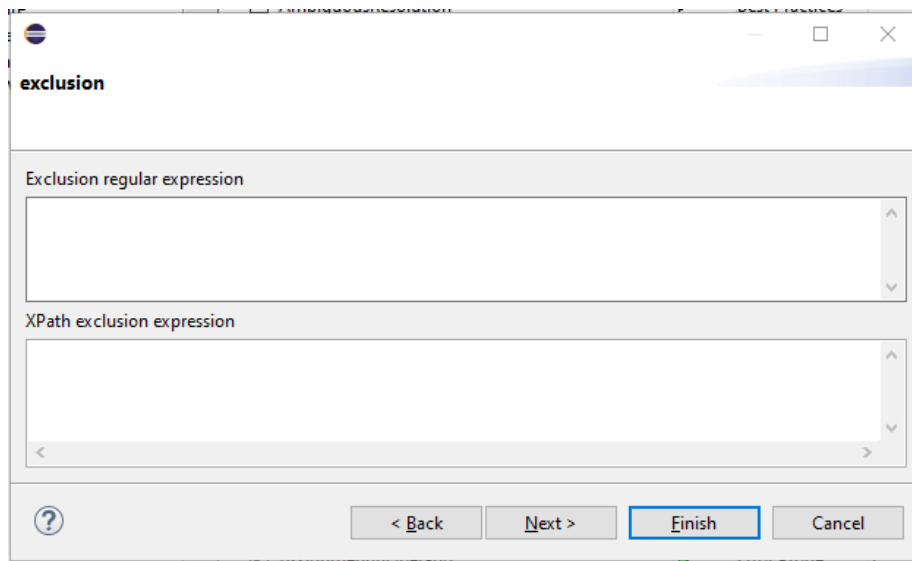


En la siguiente ventana tendremos que escribir la descripción de la regla y el mensaje que saldrá si no respetamos esa regla.



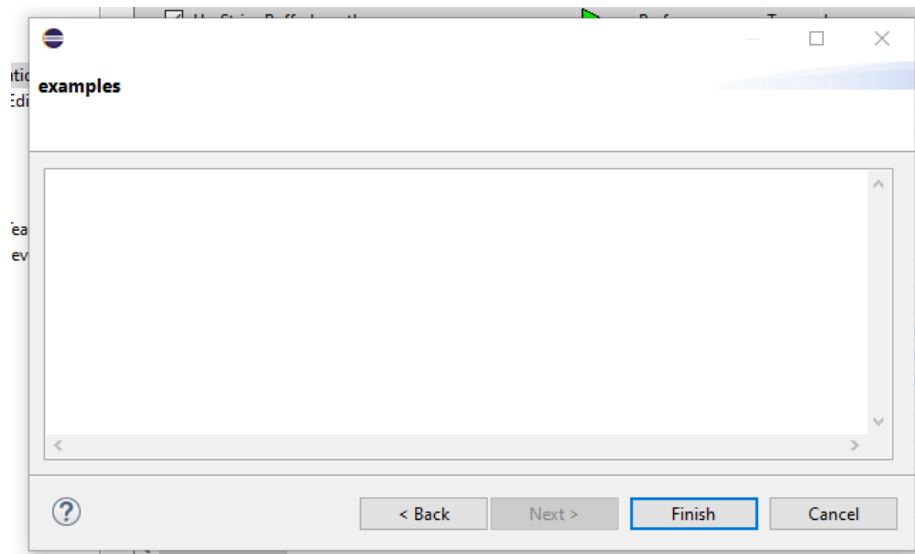
The screenshot shows a window titled "description". It contains a large text area with the text "Esta es la descripción de la regla". Below this text area is a field labeled "External Info URL:" with an "Open in Browser" button. Below that is a "Message:" field containing the text "Has incumplido una regla". At the bottom of the window are four buttons: "< Back", "Next >", "Finish", and "Cancel". The "Finish" button is highlighted with a blue border.

Nos aparecerá también una ventana para realizar exclusiones si quisiéramos hacerlo.

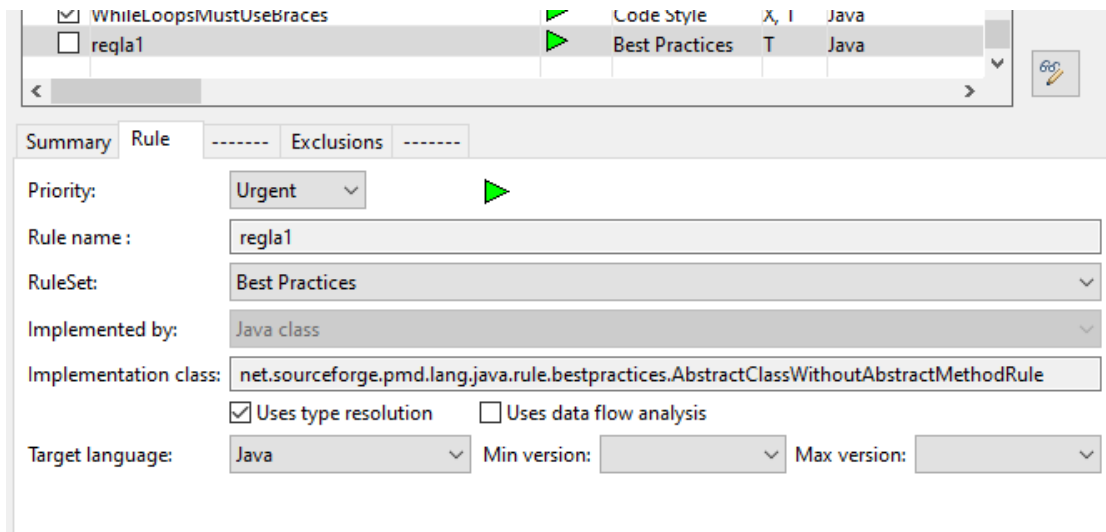


The screenshot shows a window titled "exclusion". It contains two text areas: "Exclusion regular expression" and "XPath exclusion expression". At the bottom of the window are four buttons: "< Back", "Next >", "Finish", and "Cancel". The "Finish" button is highlighted with a blue border.

Y la última ventana será una ventana para escribir un ejemplo de cómo cumplir la regla.



Y nos aparecerá entre las reglas.



## JavaDoc

En esta práctica también realizaremos actividades con JavaDoc. Como es algo que ya hemos visto y tratado con anterioridad no lo explicaré en profundidad a menos que haya algo muy importante e imprescindible que tenga que explicar para entender qué es lo que he hecho en el ejercicio.

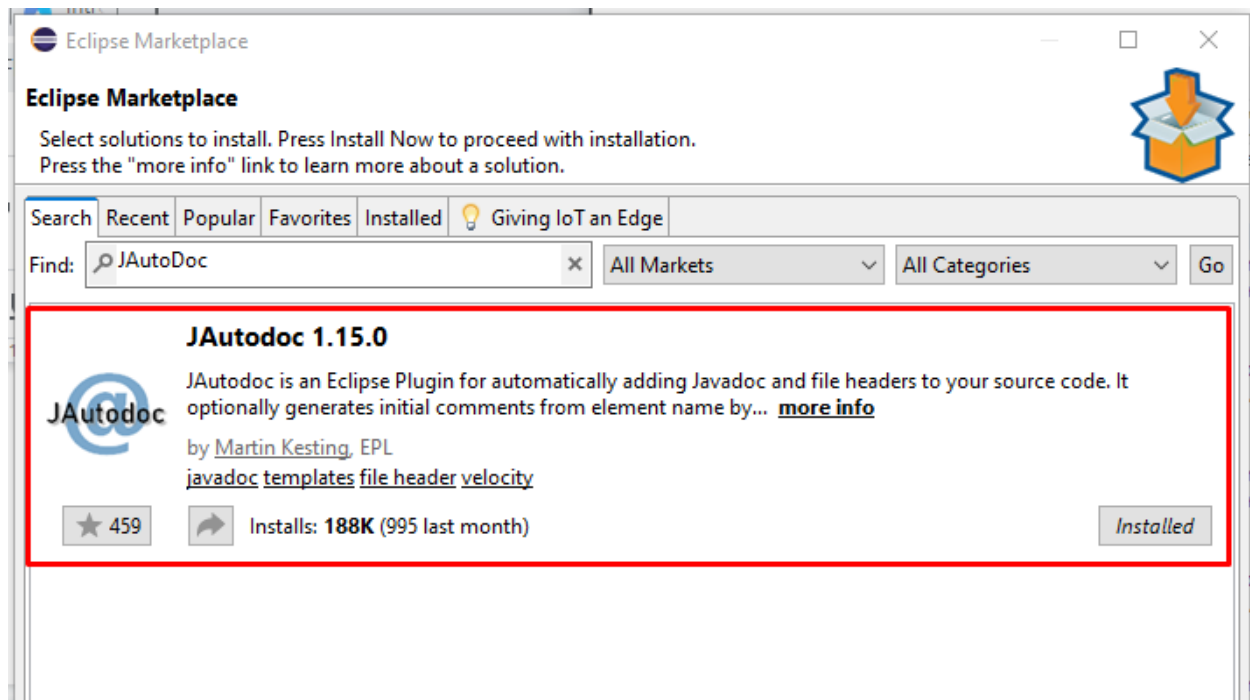
## Ejercicio 9

## Inserta comentarios Javadoc en la clase Apuesta.

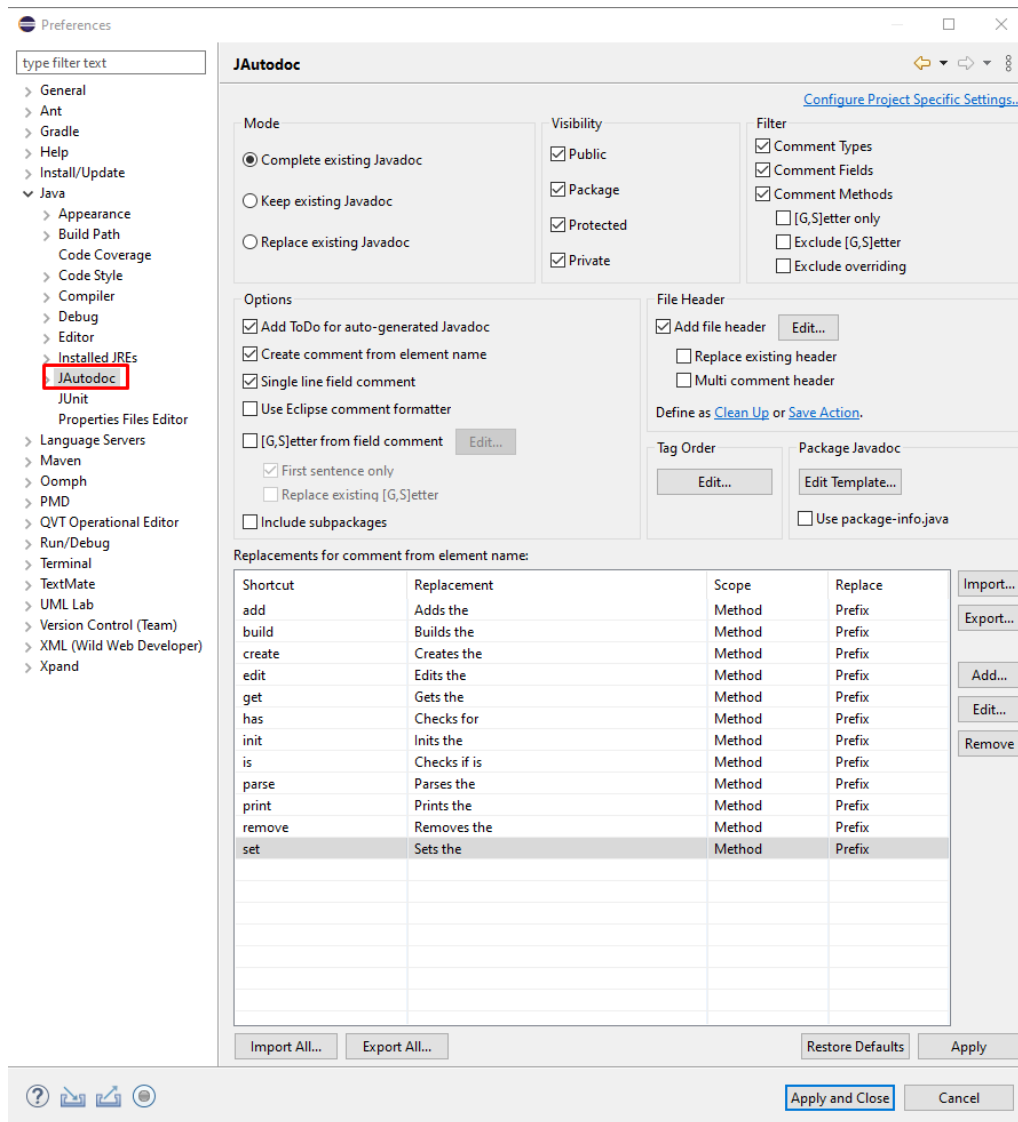
Como ya sabemos podemos escribir comentarios Javadoc como se ve en la imagen.

```
2
3 public class Apuesta {
4     /**
5      * Método get goles local
6      * @return goles_local
7      */
8     public int getGoles_local() {
9         return goles_local;
10    }
11
12    public void setGoles_local(int goles_local) {
13        this.goles_local = goles_local;
14    }
15
16    public int getGoles_visitante() {
```

Aunque también vimos que esto podíamos automatizarlo con un plugin llamado JAutoDoc, este plugin se puede configurar para personalizar los comentarios Javadoc todo lo que quieras y pulsando una combinación de botones podemos tener todo el código comentado en un segundo.



Configuración del plugin.



De esta forma comentamos todo el código.



```

1  /**
2   * @author Juan Antonio García
3   */
4   package apuesta;
5
6   // TODO: Auto-generated Javadoc
7   /**
8    * The Class Apuesta.
9    */
10  public class Apuesta {
11
12      /**
13       * Gets the goles local.
14       *
15       * @return the goles local
16       */
17      public int getGoles_local() {
18          return goles_local;
19      }
20
21      /**
22       * Sets the goles local.
23       *
24       * @param goles_local the new goles local
25       */
26      public void setGoles_local(int goles_local) {
27          this.goles_local = goles_local;
28      }
29
30      /**
31       * Gets the goles visitante.
32       *
33       * @return the goles visitante
34       */
35      public int getGoles_visitante() {
36          return goles_visitante;
37      }
38
39      /**
40       * Sets the goles visitante.
41       *
42       * @param goles_visitante the new goles visitante
43       */
44      public void setGoles_visitante(int goles_visitante) {
45          this.goles_visitante = goles_visitante;
46      }
47  }

```

```

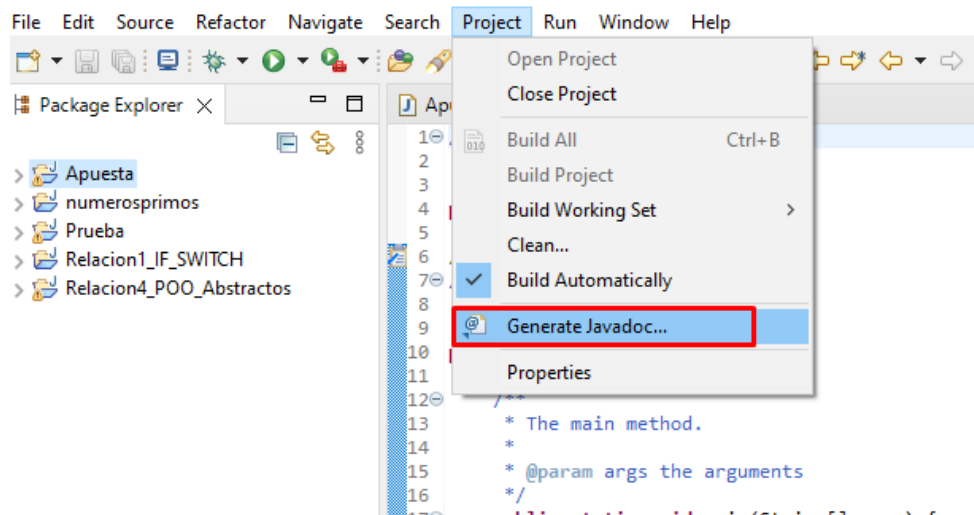
1  /**
2   * @author Juan Antonio García
3   */
4   package apuesta;
5
6   // TODO: Auto-generated Javadoc
7   /**
8    * The Class Main.
9    */
10  public class Main {
11
12      /**
13       * The main method.
14       *
15       * @param args the arguments
16       */
17      public static void main(String[] args) {
18          Apuesta laApuesta;
19          int mi_dinero;
20
21          operativa_Apuesta(0);
22      }
23
24      /**
25       * Operativa apuesta.
26       *
27       * @param dinero the dinero
28       */
29      private static void operativa_Apuesta(int dinero) {
30          Apuesta laApuesta;
31          int mi_dinero;
32          laApuesta = new Apuesta(1000, 4, 2);
33          try {
34              System.out.println("Apostando...");
35              laApuesta.apostar(25);
36          } catch (Exception e) {
37              System.out.println("Fallo al realizar la Apuesta");
38          }
39
40          try {
41              System.out.println("Intento cobrar apuesta segun el resultado del partido");
42              laApuesta.cobrar_apuesta(2, 3);
43          } catch (Exception e) {
44              System.out.println("Fallo al cobrar la apuesta");
45          }
46      }
47  }

```

## Ejercicio 10

Genera documentación Javadoc para todo el proyecto.

Una vez realizados los comentarios Javadoc generamos el Javadoc. Para esto vamos a la opción *Project* del menú superior y de ahí clicamos en la opción *Generate Javadoc*.



Elegimos de qué proyecto generar el Javadoc, la visibilidad del Javadoc, donde se guardará el Javadoc, etc...

Una vez terminado tendremos nuestro archivo html con la información de nuestro proyecto. La Documentación Javadoc se adjuntará junto con esta práctica **en una carpeta llamada doc dentro de la carpeta Apuestas**.

