Entornos del desarrollo
Pedro Carmona Mallén

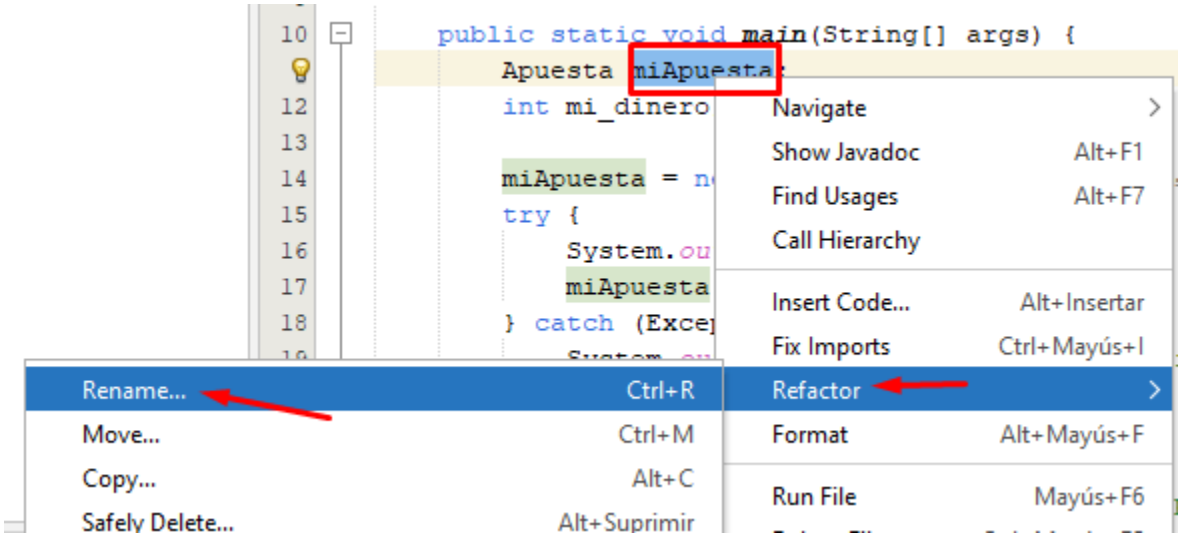
Trabajo ENDE.

4 de marzo del 2023

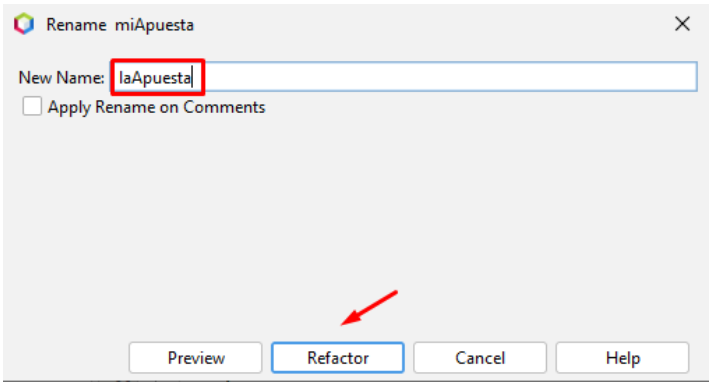
1.- REFACTORIZACIÓN.

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

- Seleccionamos la variable que queremos cambiar, en nuestro caso la variable “miApuesta”, la cuál se encuentra en la clase “Main”. Una vez seleccionada,haremos click derecho sobre nuestra variable y accederemos a la opción “Refactor➔Rename...”.



- Una vez aquí , nos aparecerá para cambiar el nombre de la variable, le ponemos el nombre que le queremos dar a dicha variable, y le damos a “Refactor” y ya nos aparecerá, todos los lugares donde la variable ha sido cambiada. Véase aquí:



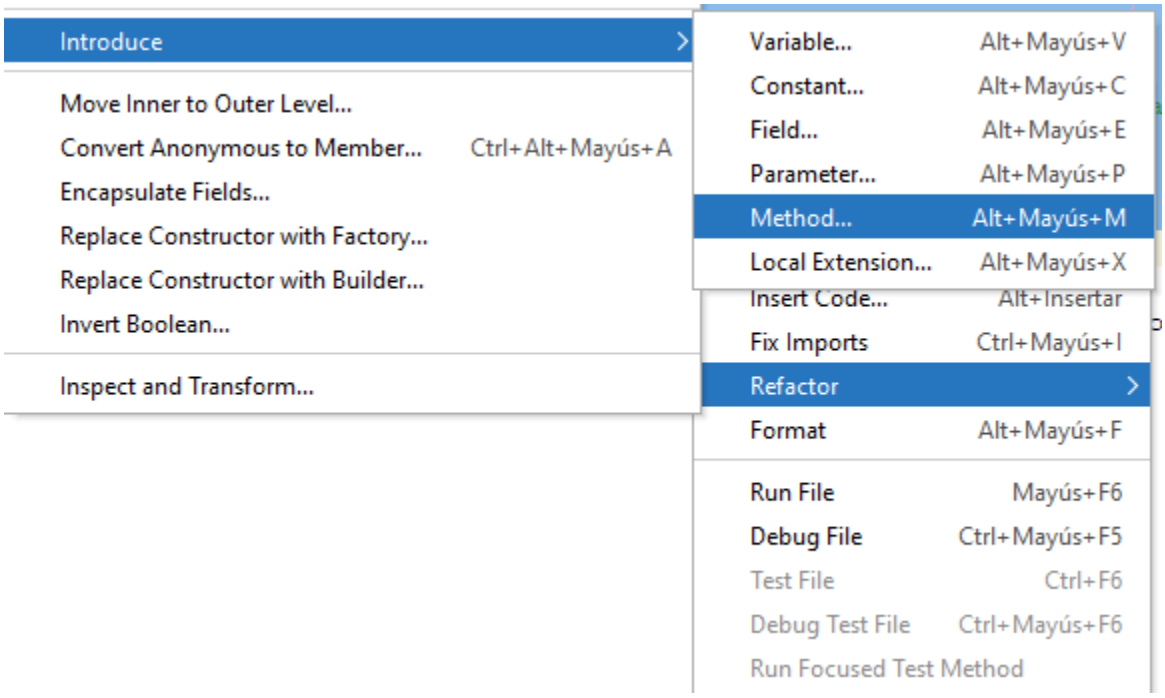

```
public static void main(String[] args) {
    Apuesta laApuesta;
    int mi_dinero;

    laApuesta = new Apuesta(dinero_disp:1000, g
    try {
        System.out.println(x: "Apostando...")
        laApuesta.apostar(dinero: 25);
    } catch (Exception e) {
        System.out.println(x: "Fallo al reali
    }

    try {
        System.out.println(x: "Intento cobrar
        laApuesta.cobrar_apuesta(cantidad_goles
    } catch (Exception e) {
        System.out.println(x: "Fallo al cobra
    }
    mi_dinero = laApuesta.getDinero_disp();
    System.out.println("El dinero que tengo
}
```

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

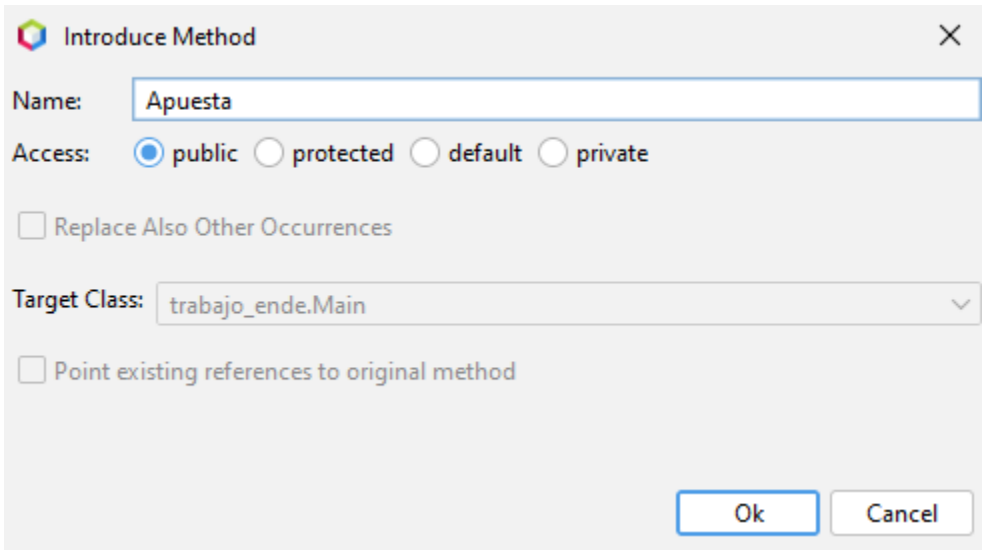
- Seleccione las operaciones de Apuesta que se realizan con “laApuesta” y que quiero englobarlas en un método.

```
public static void main(String[] args) {
    Apuesta laApuesta;
    int mi_dinero;

    laApuesta = new Apuesta(dinero_disp:1000, goles_local:4, goles_visitante:2);
    try {
        System.out.println(x: "Apostando...");
        laApuesta.apostar(dinero: 25);
    } catch (Exception e) {
        System.out.println(x: "Fallo al realizar la Apuesta");
    }

    try {
        System.out.println(x: "Intento cobrar apuesta segun el resultado del partido");
        laApuesta.cobrar_apuesta(cantidad_goles_local: 2, cantidad_goles_visit: 3);
    } catch (Exception e) {
        System.out.println(x: "Fallo al cobrar la apuesta");
    }
```

- Una vez hecho esto, volveremos a hacer click derecho sobre él, y como hemos echo antes, nos iremos a “Refactor” y ahí nos iremos a la opción de “Introduce” , y dentro de esta, seleccionaremos la opción “Method”. Véase aquí:



- Una vez dentro, le daremos el nombre de “Apuesta” al nuevo método operativo que vamos a crear con esta opción:

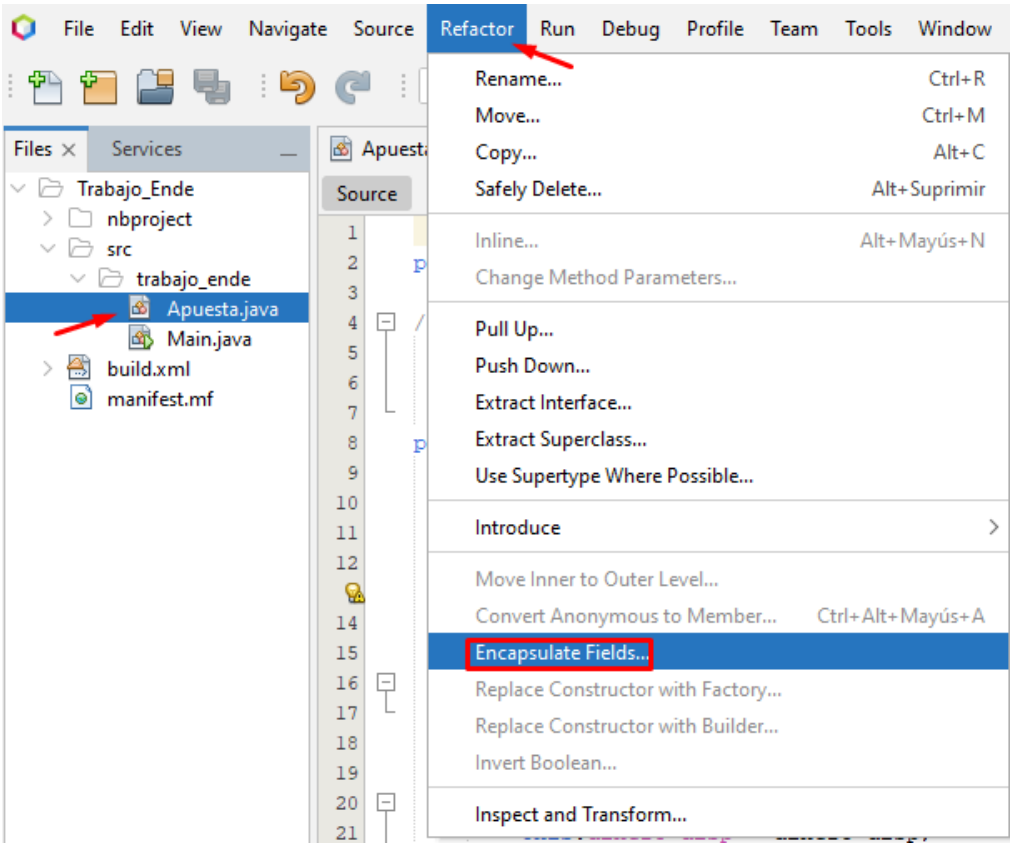


- Y este sería el resultado final:

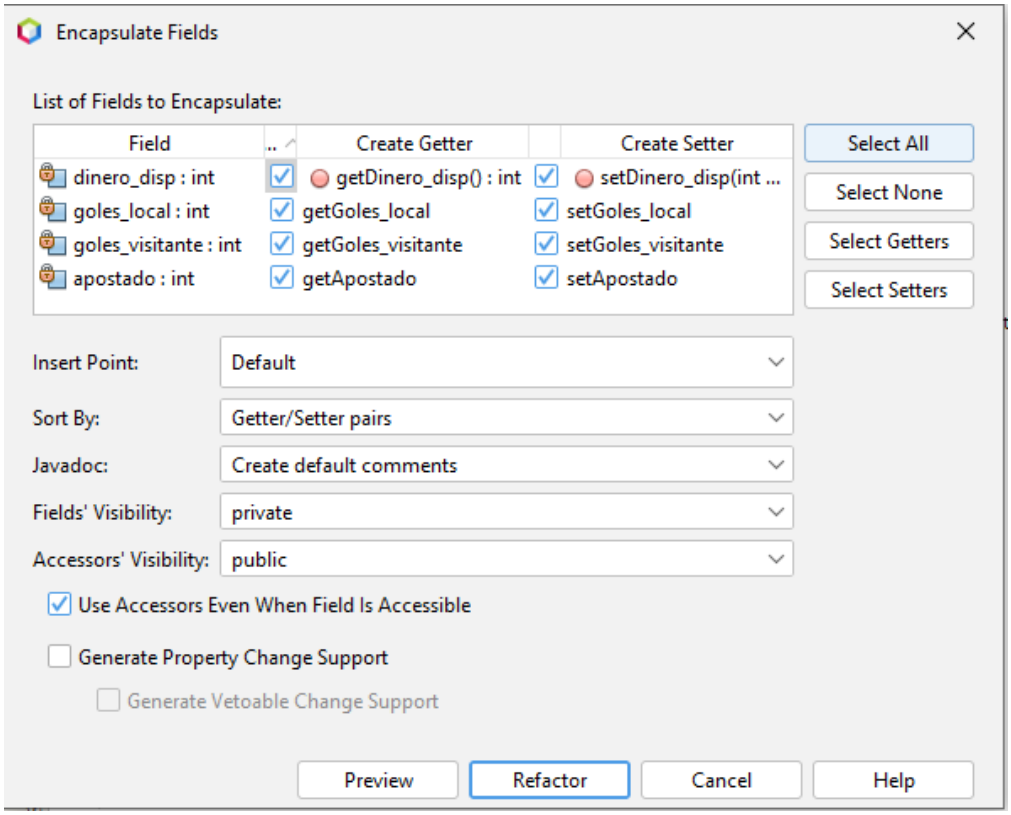
```
    public static void Apuesta(Apuesta laApuesta) {
        try {
            System.out.println(x: "Apostando...");
            laApuesta.apostar(dinero: 25);
        } catch (Exception e) {
            System.out.println(x: "Fallo al realizar la Apuesta");
        }

        try {
            System.out.println(x: "Intento cobrar apuesta segun el resultado del partido");
            laApuesta.cobrar_apuesta(cantidad_goles_local: 2, cantidad_goles_visit: 3);
        } catch (Exception e) {
            System.out.println(x: "Fallo al cobrar la apuesta");
        }
    }

}
```

1.3.- Encapsula todos los atributos de la clase Apuesta.

- A continuación, vamos a encapsular los atributos de la clase apuesta. Para ellos clickeamos sobre la clase apuesta y nos iremos al menú superior y accederemos en “Refactor” a la opción de “Encapsule Fields”:



- En la ventana que nos aparece, seleccionamos los siguientes campos:



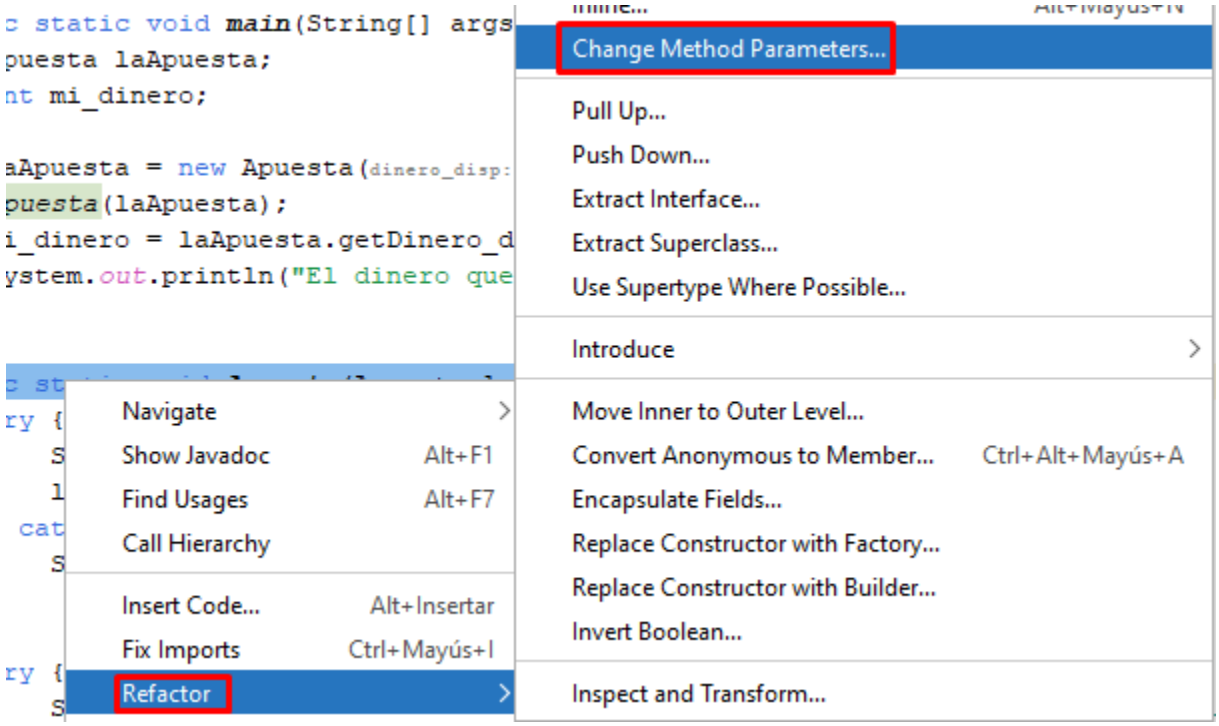
- Pulsamos en “Refactor” y como podremos ver ya se habrán creado todas las opciones seleccionadas:

```
public class Apuesta {

    /**
     * @return the goles_local
     */
    public int getGoles_local() {
        return goles_local;
    }

    /**
     * @param goles_local the goles_local to set
     */
    public void setGoles_local(int goles_local) {
        this.goles_local = goles_local;
    }

    /**
     * @return the goles_visitante
     */
    public int getGoles_visitante() {
        return goles_visitante;
    }
```

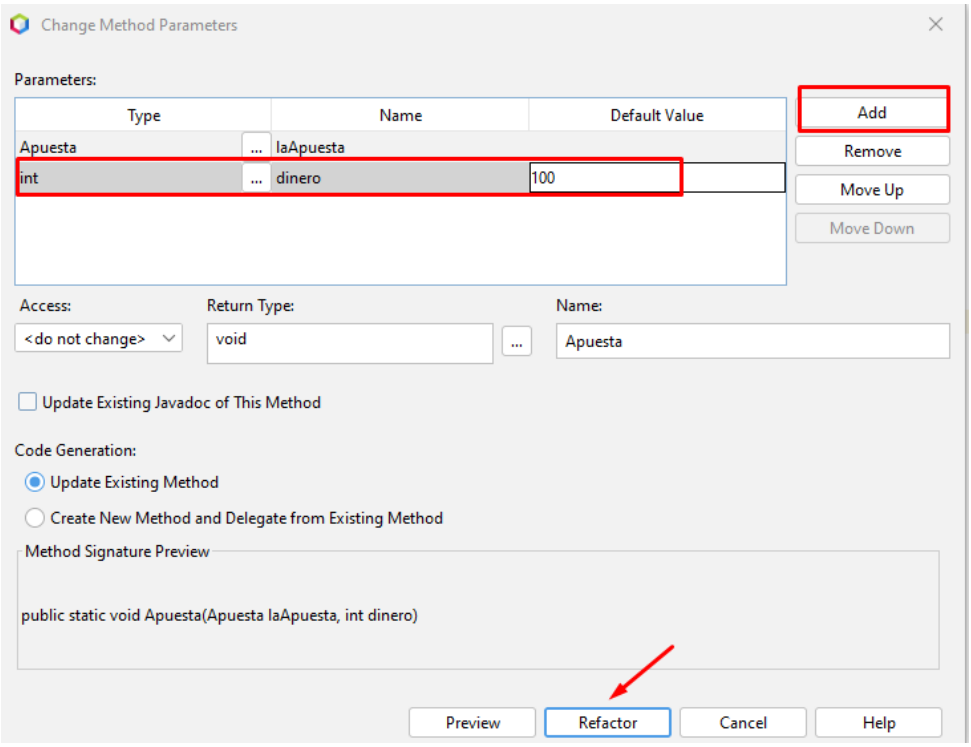
(ANOTACIÓN: Aparecen más pero para que no fuese una captura tan grande, he mostrado sólo eso, para que se vea que ha funcionado.)

1.4.- Añadir un nuevo parámetro al método “laApuesta”, de nombre dinero y de tipo int.

- Seleccionamos nuestro método “laApuesta” y hacemos click derecho sobre el, accedemos a “Refactor” y en el a la opción de “Change Method Parameters….”:



- Una vez aquí, le damos a la opción de “Add” para añadir un nuevo parametro, este sera de tipo entero, de nombre “dinero” y de valor por defecto 100. Una vez todo escrito le damos a “Refactor”:



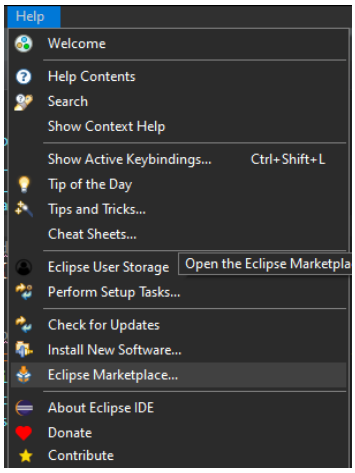
- Y como podremos observar, los parámetros del método, contienen ahora el parámetro que le hemos añadido:

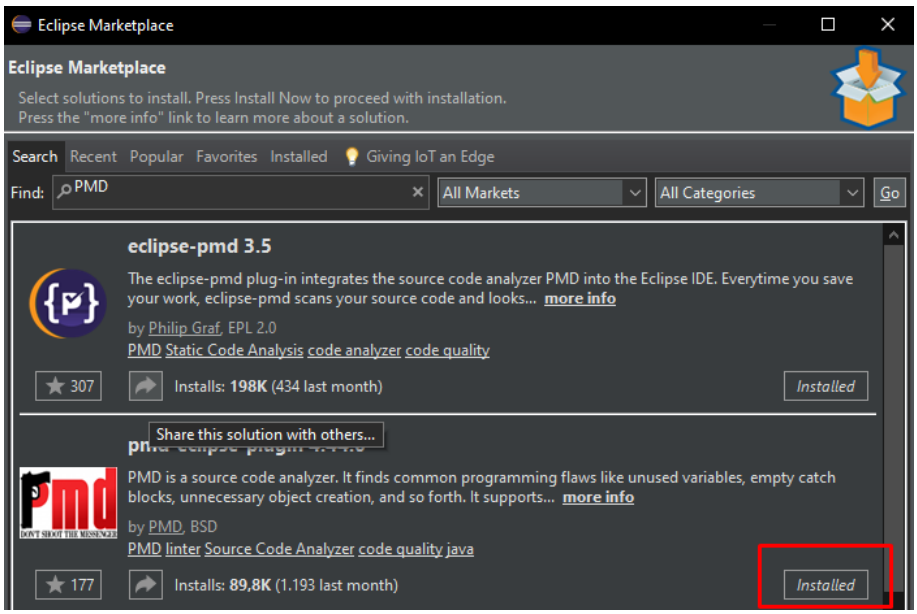
```
public static void Apuesta(Apuesta laApuesta, int dinero) {
    try {
        System.out.println(x: "Apostando...");
        laApuesta.apostar(dinero: 25);
    } catch (Exception e) {
        System.out.println(x: "Fallo al realizar la Apuesta");
    }
```

2. ANALIZADOR DE CÓDIGO.

2.1.- Descarga e instala el plugin PMD.

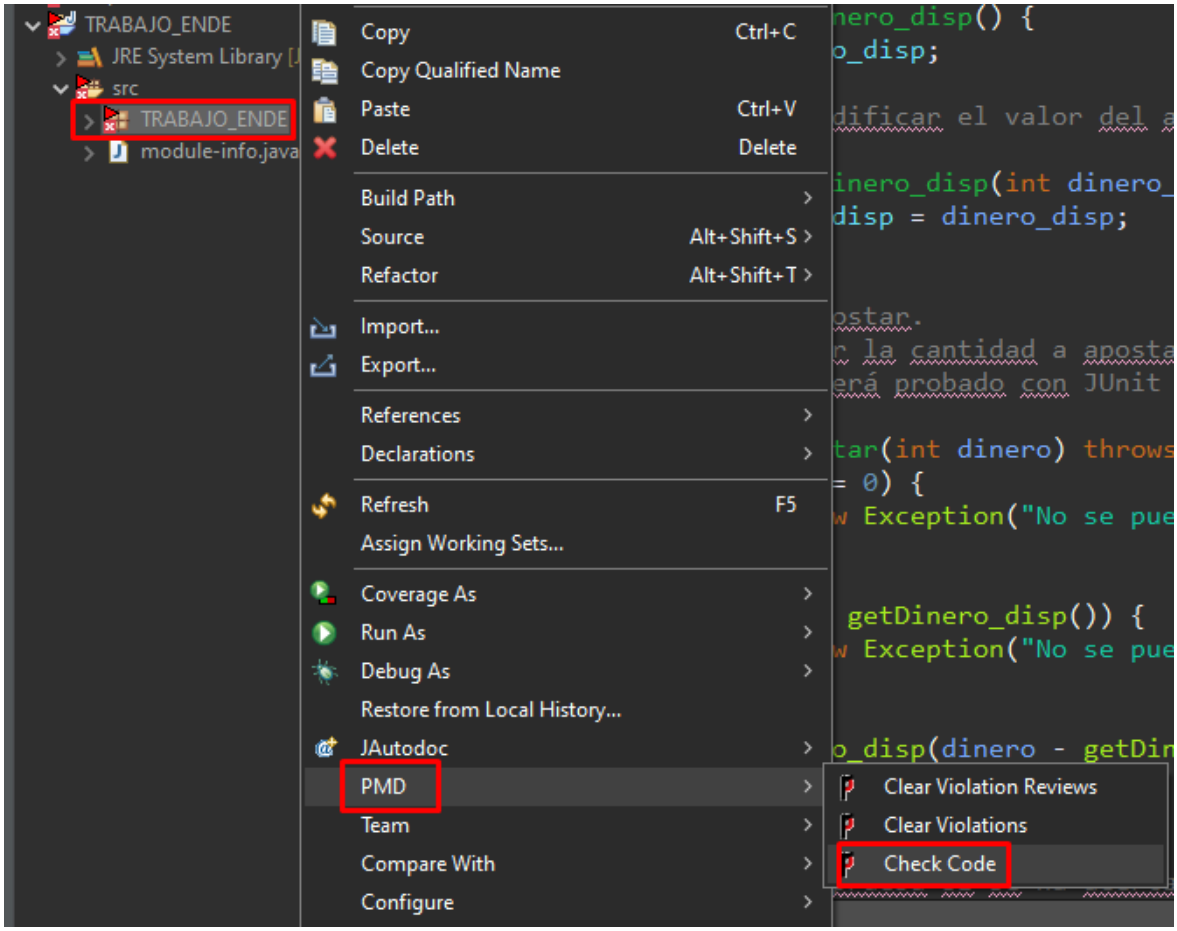
- “PMD es una herramienta de calidad de código encargada de validar los estándares de construcción de un desarrollo. Es decir, chequea la sintaxis del código fuente que ha sido desarrollado, encontrando las ocurrencias de un determinado problema que haya sido previamente configurado para ser detectado.”
- Primero vamos a instalar PMD en Eclipse, para ello accederemos a “Eclipse Marketplace” el cual se encuentra en el “Help” de nuestro menú superior. Dentro de él buscamos “PMD” y le damos a Instalar. Véase aquí:



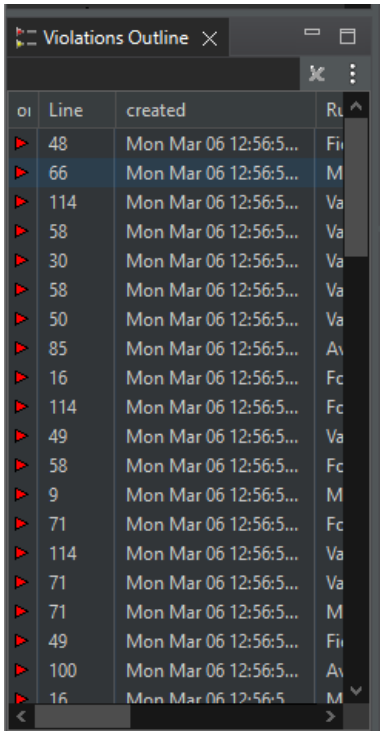


2.2.- Ejecuta el analizador de código PMD.

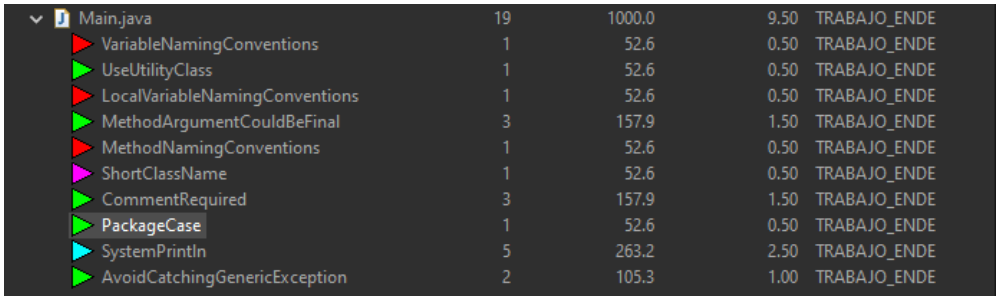
- A continuación, vamos a ejecutar el analizador de código PMD sobre nuestro paquete con nuestras clases. Para ello, hacemos click derecho sobre nuestro paquete “TRABAJO_ENDE” y le damos a la opción de PMD y en ella a la opción “Check Code”:



- Una vez hecho esto, nos apareceran las siguientes ventanas:

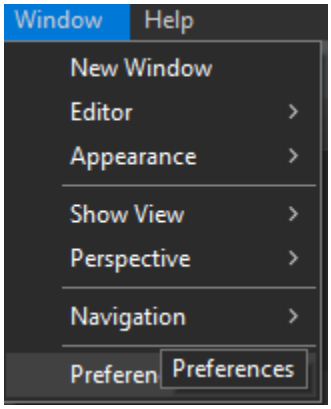

Violations Outline
Line
created
48 Mon Mar 06 12:56:5...
66 Mon Mar 06 12:56:5...
114 Mon Mar 06 12:56:5...
58 Mon Mar 06 12:56:5...
30 Mon Mar 06 12:56:5...
58 Mon Mar 06 12:56:5...
50 Mon Mar 06 12:56:5...
85 Mon Mar 06 12:56:5...
16 Mon Mar 06 12:56:5...
114 Mon Mar 06 12:56:5...
49 Mon Mar 06 12:56:5...
58 Mon Mar 06 12:56:5...
9 Mon Mar 06 12:56:5...
71 Mon Mar 06 12:56:5...
114 Mon Mar 06 12:56:5...
71 Mon Mar 06 12:56:5...
71 Mon Mar 06 12:56:5...
49 Mon Mar 06 12:56:5...
100 Mon Mar 06 12:56:5...


TRABAJO_ENDE 96 1476.9 6.40 TRABAJO_ENDE
Apuesta.java 77 1673.9 5.92 TRABAJO_ENDE
SimplifyBooleanExpressions 1 21.7 0.08 TRABAJO_ENDE
MethodArgumentCouldBeFinal 12 260.9 0.92 TRABAJO_ENDE
LongVariable 2 43.5 0.15 TRABAJO_ENDE
CommentRequired 9 195.7 0.69 TRABAJO_ENDE
DefaultPackage 1 21.7 0.08 TRABAJO_ENDE
FormalParameterNamingConventio 8 173.9 0.62 TRABAJO_ENDE
CommentSize 3 65.2 0.23 TRABAJO_ENDE
SignatureDeclareThrowsException 3 65.2 0.23 TRABAJO_ENDE
AvoidThrowingRawExceptionTypes 4 87.0 0.31 TRABAJO_ENDE
VariableNamingConventions 11 239.1 0.85 TRABAJO_ENDE
UselessParentheses 1 21.7 0.08 TRABAJO_ENDE
MethodNamingConventions 8 173.9 0.62 TRABAJO_ENDE
DataClass 1 21.7 0.08 TRABAJO_ENDE

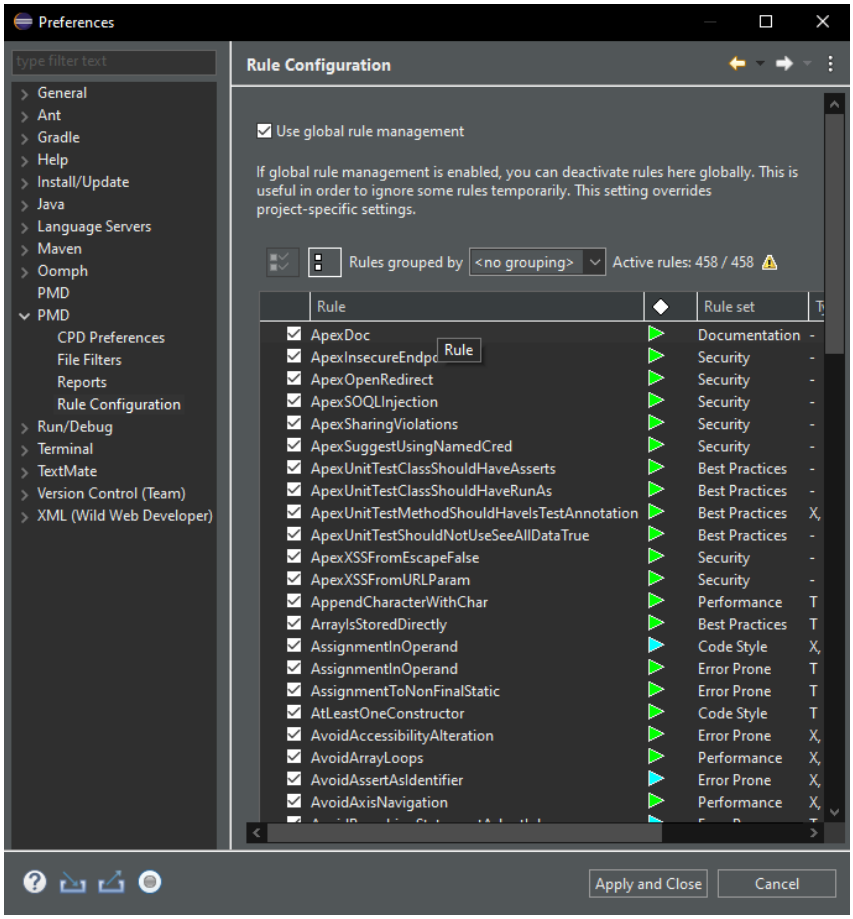

Main.java 19 1000.0 9.50 TRABAJO_ENDE
VariableNamingConventions 1 52.6 0.50 TRABAJO_ENDE
UseUtilityClass 1 52.6 0.50 TRABAJO_ENDE
LocalVariableNamingConventions 1 52.6 0.50 TRABAJO_ENDE
MethodArgumentCouldBeFinal 3 157.9 1.50 TRABAJO_ENDE
MethodNamingConventions 1 52.6 0.50 TRABAJO_ENDE
ShortClassName 1 52.6 0.50 TRABAJO_ENDE
CommentRequired 3 157.9 1.50 TRABAJO_ENDE
PackageCase 1 52.6 0.50 TRABAJO_ENDE
SystemPrintln 5 263.2 2.50 TRABAJO_ENDE
AvoidCatchingGenericException 2 105.3 1.00 TRABAJO_ENDE

2.3.- Añade tres reglas nuevas al analizador de PMD y comenta su utilidad.

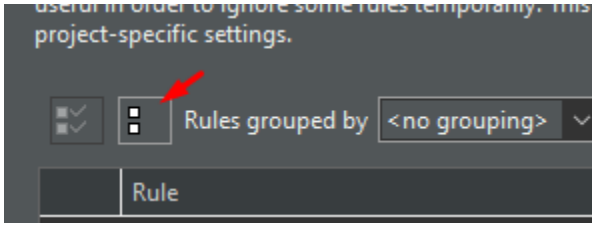
- Para poder hacer esto, accedemos a la opción de nuestro menú superior “Window” y en ella a la opción de “Preferences”:



- Una vez dentro de estas, no iremos a “PMD” y dentro de este, a la opción de “Rule Configuration” y nos aparecerá lo siguiente:



- Una vez aquí, pulsaremos el siguiente símbolo, para desmarcar todas las reglas y elegir tres de ellas que queramos:

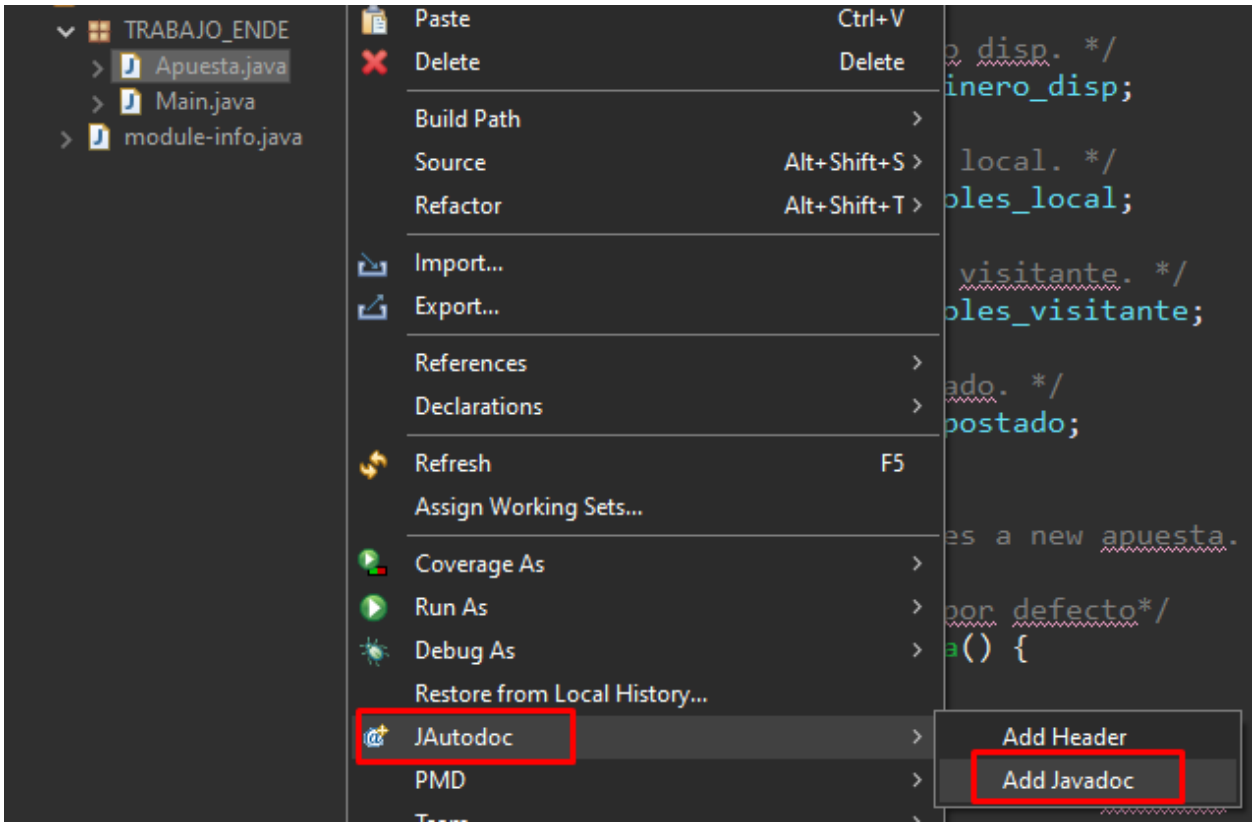


- Las reglas que vamos a elegir son las siguientes:
    - “AddEmptyString”:La conversión de literales a cadenas al concatenarlas con cadenas vacías es ineficiente. Es mucho mejor usar uno de los métodos específicos de tipo `toString()` en su lugar o `String.valueOf()`.
    - “ArrayIsStoredDirectly”: Los constructores y métodos que reciben matrices deben clonar objetos y almacenar la copia. Esto evita que los cambios futuros del usuario afecten la matriz original.
    - “CheckResultSet”: Compruebe siempre los valores de retorno de los métodos de navegación (siguiente, anterior, primero, último) de un ResultSet. Si el valor devuelto es 'falso', debe manejarse correctamente.

3. JAVADOC.

3.1.- Inserta comentarios Javadoc en la clase Apuesta.

- Para generar el “Javadoc” en nuestra clase apuesta, lo haremos a través del “JAutodoc”. Para ello, haremos click derecho sobre nuestra clase “Apuesta” y en ella la opción de “JAutodoc” y dentro de este a la opción de “Add Javadoc”:



- Y nos aparecerá los siguientes comentarios de Javadoc en nuestra clase “Apuesta”:

```java
package TRABAJO_ENDE;

// TODO: Auto-generated Javadoc
/**
 * The Class Apuesta.
 */
/**
 *
 * @author Pedro Carmona Mallén
 * @version Version final del documento.
 */
public class Apuesta {

    /**
     * Gets the goles local.
     *
     * @return the goles_local
     */

    public int getGoles_local() {
        return goles_local;
    }

    /**
     * Sets the goles local.
     *
     * @param goles_local the goles_local to set
     */
    public void setGoles_local(int goles_local) {
        this.goles_local = goles_local;
    }

    /**
     * Gets the goles visitante.
     *
     * @return the goles_visitante
     */
    public int getGoles_visitante() {
        return goles_visitante;
    }

    /**
     * Sets the goles visitante.
     *
     * @param goles_visitante the goles_visitante to set
     */
```

```java
    public void setGoles_visitante(int goles_visitante) {
        this.goles_visitante = goles_visitante;
    }

    /**
     * Gets the apostado.
     *
     * @return the apostado
     */
    public int getApostado() {
        return apostado;
    }

    /**
     * Sets the apostado.
     *
     * @param apostado the apostado to set
     */
    public void setApostado(int apostado) {
        this.apostado = apostado;
    }

    /** The dinero_disp. */
    private int dinero_disp;

    /** The goles_local. */
    private int goles_local;

    /** The goles_visitante. */
    private int goles_visitante;

    /** The apostado. */
    private int apostado;

    /**
     * Instantiates a new apuesta.
     */
    /*Contructor por defecto*/
    public Apuesta() {
    }

    /**
     * Instantiates a new apuesta.
     *
     * @param dinero_disp the dinero_disp
     * @param goles_local the goles_local
     * @param goles_visitante the goles_visitante
```

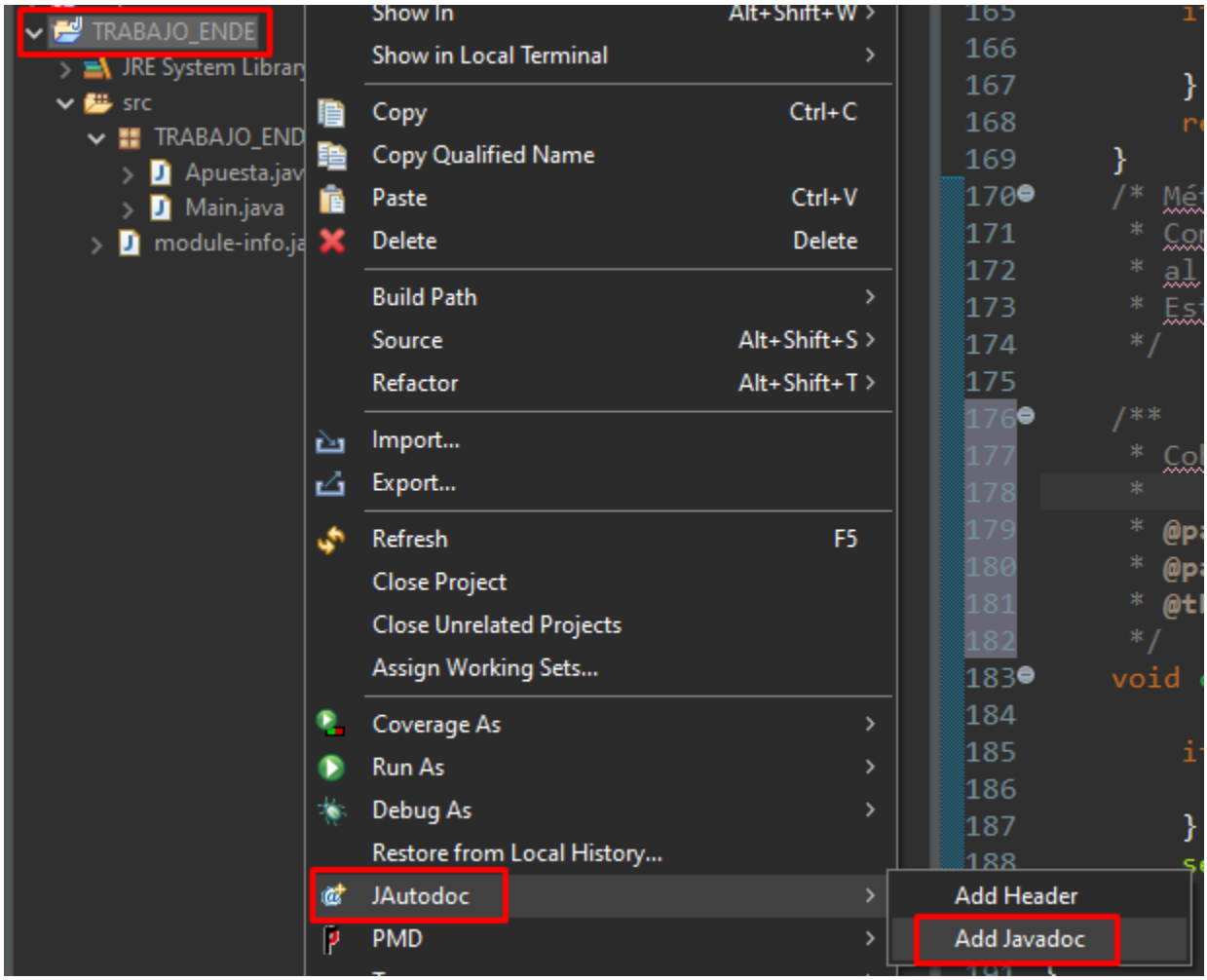
```java
     */
    /*Contructor por parámetros*/
    public Apuesta(int dinero_disp, int goles_local, int goles_visitante) {
        this.dinero_disp = dinero_disp;
        this.goles_local = goles_local;
        this.goles_visitante = goles_visitante;
        this.apostado = 0;
    }
    /*Método para obtener el valor del atributo dinero_disp*/

    /**
     * Gets the dinero disp.
     *
     * @return the dinero disp
     */
    public int getDinero_disp() {
        return dinero_disp;
    }
    /*Método para modificar el valor del atributo dinero_disp*/

    /**
     * Sets the dinero disp.
     *
     * @param dinero_disp the new dinero disp
     */
    public void setDinero_disp(int dinero_disp) {
        this.dinero_disp = dinero_disp;
    }

    /**
     * Apostar.
     *
     * @param dinero the dinero
     * @throws Exception the exception
     */
    /*Método para apostar.
     * Permite elegir la cantidad a apostar, no pudiendo ser inferior a 1 ni superior a tu saldo disponible
     * Este método será probado con JUnit
     */
    public void apostar(int dinero) throws Exception {
        if (dinero <= 0) {
            throw new Exception("No se puede apostar menos de 1€");
        }

        if (dinero > getDinero_disp()) {
            throw new Exception("No se puede apostar mas de lo que tienes");
        }
```

```
141         {
142             setDinero_disp(dinero - getDinero_disp());
143             setApostado(dinero);
144         }
145     }
146     /*Método que comprueba si se ha acertado el resultado del partido
147      * En caso de que lo haya acertado devuelve true. Chequea que no se metan menos de 0 goles
148      *
149      */
150 
151     /**
152      * Comprobar resultado.
153      *
154      * @param local the local
155      * @param visitante the visitante
156      * @return true, if successful
157      * @throws Exception the exception
158      */
159     public boolean comprobar_resultado(int local, int visitante) throws Exception {
160         boolean acertado = false;
161         if ((local < 0) || (visitante) < 0) {
162             throw new Exception("Un equipo no puede meter menos de 0 goles, por malo que sea");
163         }
164 
165         if (getGoles_local() == local && getGoles_visitante() == visitante) {
166             acertado = true;
167         }
168         return acertado;
169     }
170     /* Método para cobrar la apuesta.
171      * Comprueba que se acertó el resultado y, en ese caso, añade el valor apostado multiplicado por 10
172      * al saldo disponible
173      * Este método se va a probar con Junit
174      */
175 
176     /**
177      * Cobrar apuesta.
178      *
179      * @param cantidad_goles_local the cantidad goles local
180      * @param cantidad_goles_visit the cantidad goles visit
181      * @throws Exception the exception
182      */
183     void cobrar_apuesta(int cantidad_goles_local, int cantidad_goles_visit) throws Exception {
184 
185         if (comprobar_resultado(cantidad_goles_local, cantidad_goles_visit) == false) {
186             throw new Exception("No se puede cobrar una apuesta no acertada");
```

```
187             }
188             setDinero_disp(getDinero_disp() * 10);
189 
190         }
191 }
192 
193 
```

3.2.- Genera documentación Javadoc para todo el proyecto.

- Para hacer esto, haremos click derecho sobre el proyecto donde se encuentra nuestras dos clases, al cuál le hemos dado el nombre de “TRABAJO_ENDE” y aqui volveremos hacer lo que hemos hecho anteriormente para el de la clase “Apuesta”, le daremos a la opción de “JAutodoc”:



- Y como podemos ver, en nuestra clase “Main” también se ha generado el Javadoc:

```java
package TRABAJO_ENDE;
// TODO: Auto-generated Javadoc
/**
 * The Class Main.
 */
/**
 * @author Pedro Carmona Mallén
 * @version Version final del documento.
 */
public class Main {
	/**
	 * The main method.
	 *
	 * @param args the arguments
	 */
	public static void main(String[] args) {
		Apuesta laApuesta;
		int mi_dinero;

		laApuesta = new Apuesta(1000, 4, 2);
		Apuesta(laApuesta, 100);
		mi_dinero = laApuesta.getDinero_disp();
		System.out.println("El dinero que tengo tras las apuestas es " + mi_dinero);
	}
	/**
	 * Apuesta.
	 *
	 * @param laApuesta the la apuesta
	 * @param dinero the dinero
	 */
	public static void Apuesta(Apuesta laApuesta, int dinero) {
		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");
		}
	}
}

```