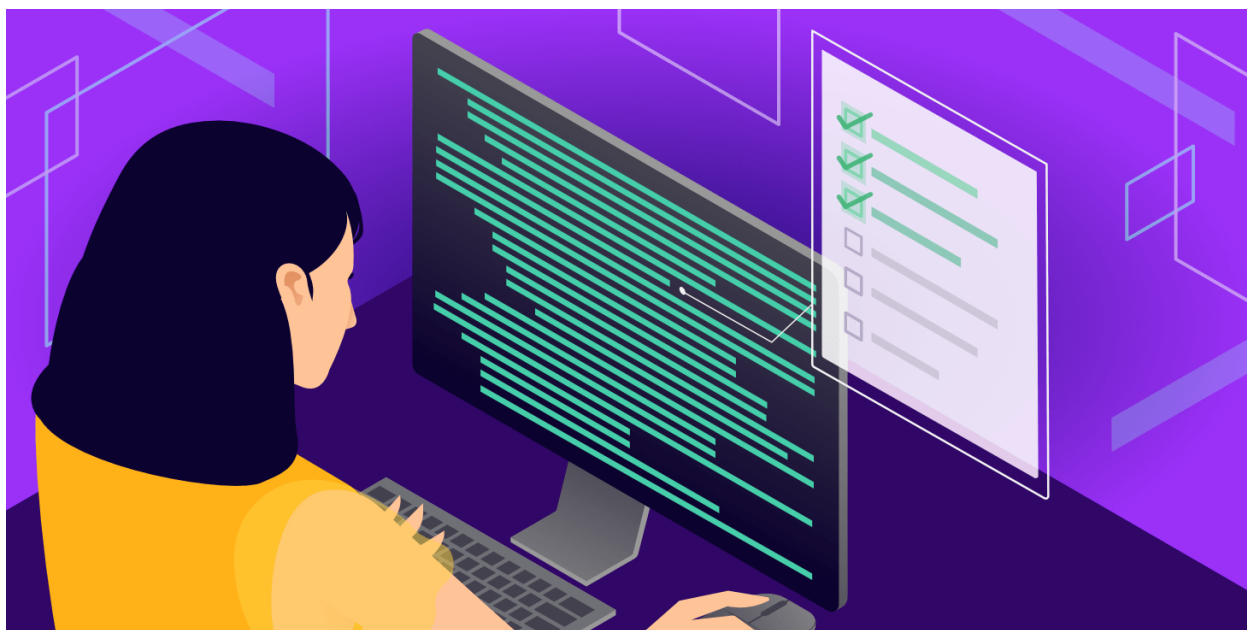


TEMA 4

OPTIMIZACIÓN Y DOCUMENTACIÓN

TAREA 1



Francisco Fernández-Pro Barbarroja

1º D.A.M.

ÍNDICE

1. INTRODUCCIÓN	3
2. REFACTORIZACIÓN	4
2.1. Cambio de nombre a la variable “miApuesta” por “laApuesta”.	4
2.2. Método operativa Apuesta.	6
2.3. Encapsulamiento de los atributos de la clase Apuesta	10
2. 4. Añadir un nuevo parámetro al método operativa_Apuesta, de nombre dinero y de tipo int.	12
3. ANALIZADOR DE CÓDIGO	15
3.2. Configuración del analizador de código PMD: escaneado automático del proyecto, a intervalos regulares y adicción de tres nuevas reglas	20
4. JAVADOC	22
4.1. Inserción de comentarios para el JavaDoc.	22
4.2. Generación de documentación JavaDoc del proyecto Apuesta en Eclipse.	24
5. GitHub y entrega del documento	28
5.1. Creación de Repositorio Público en GitHub.com	28
5.2. Conexión GIT con nuestro repositorio currofpb/OptimizaciónDocumentacion.	30
5.2. Conectar repositorio local con repositorio remoto.	32
5.3. Agregar primera rama al repositorio remoto desde nuestro repositorio local: archivos java Apuesta (código original y actualizado).	33
5.4. Creación de una segunda rama para subir los archivos pdf y documentación Javadoc.	37
5.5. Unificación de rama Master con rama rama_texto en repositorio local (opcional).	39
5.6. Demostración de ramas en GitHub.	40
6. Bibliografía	43

1. INTRODUCCIÓN

En el proyecto Java que se adjunta (Apuesta), hay definida una **Clase Apuesta**, que cuenta con una serie de atributos y métodos. El proyecto cuenta asimismo con una **Clase Main**, donde se hace uso de la clase descrita. Basándonos en ese proyecto, vamos a realizar varias actividades referentes a la **optimización** y la **documentación** de proyectos.

Para la optimización de este proyecto, realizaremos una **refactorización** del código presentado, SEGUIR

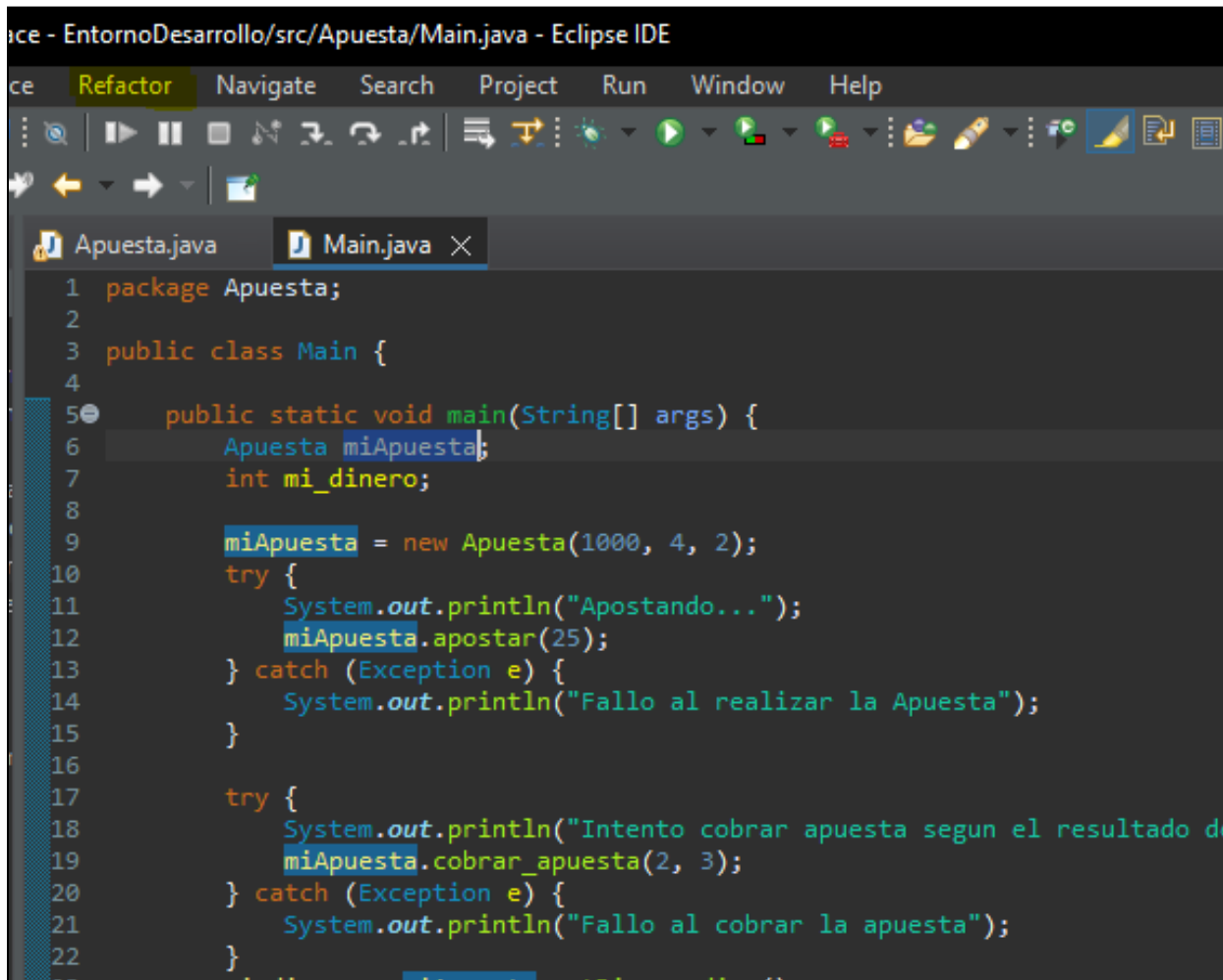
Aplicada la optimización, **analizaremos el código** para encontrar fallas de programación comunes que puedan seguir existiendo, con ayuda de un plugin llamado PMD, el cual desarrollaremos más adelante.

Por último, en el apartado de la documentación, generaremos comentarios **Javadoc**, y con ello, toda la documentación esencial del proyecto.

2. REFACTORIZACIÓN

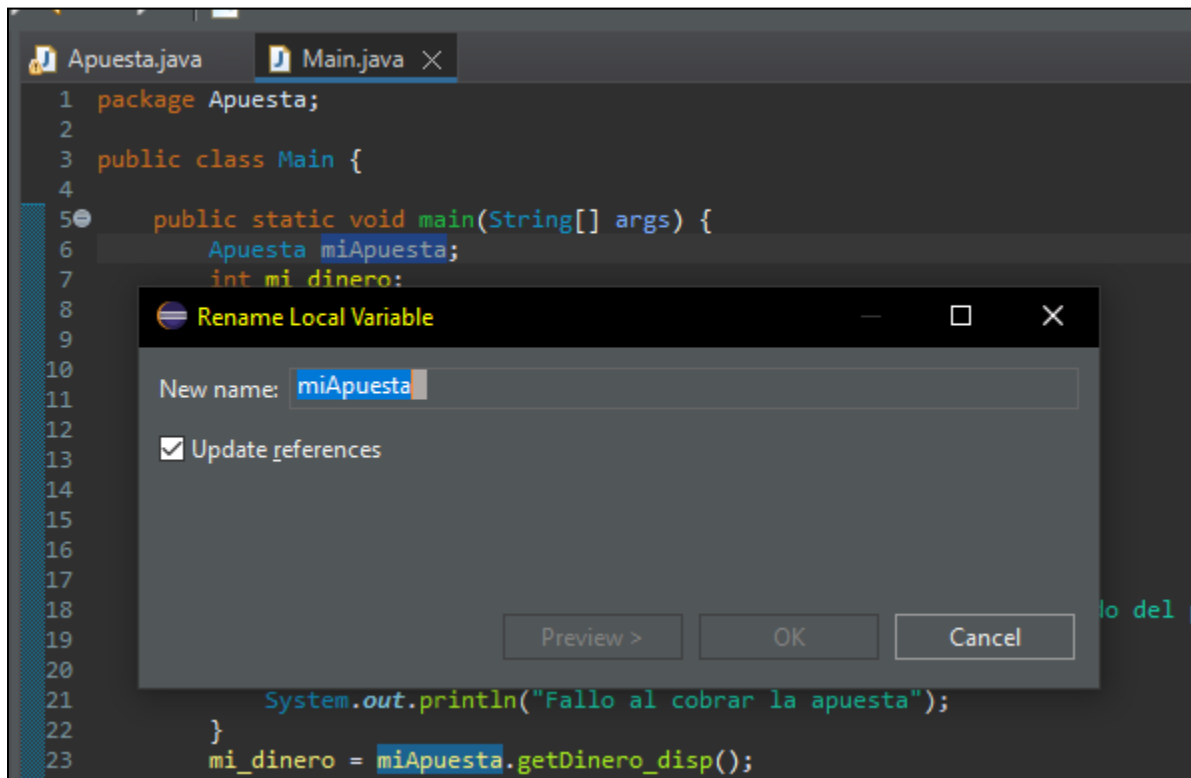
2.1. Cambio de nombre a la variable “miApuesta” por “laApuesta”.

Para cambiar o renombrar el nombre de una variable en el programa Eclipse, tenemos que clicar dos veces la variable, en este caso “miApuesta”, en la clase donde se encuentre, y en la parte superior, en las opciones presentadas por el programa, clicamos “Refactor” < “rename”, y renombramos la variable por “laApuesta”.

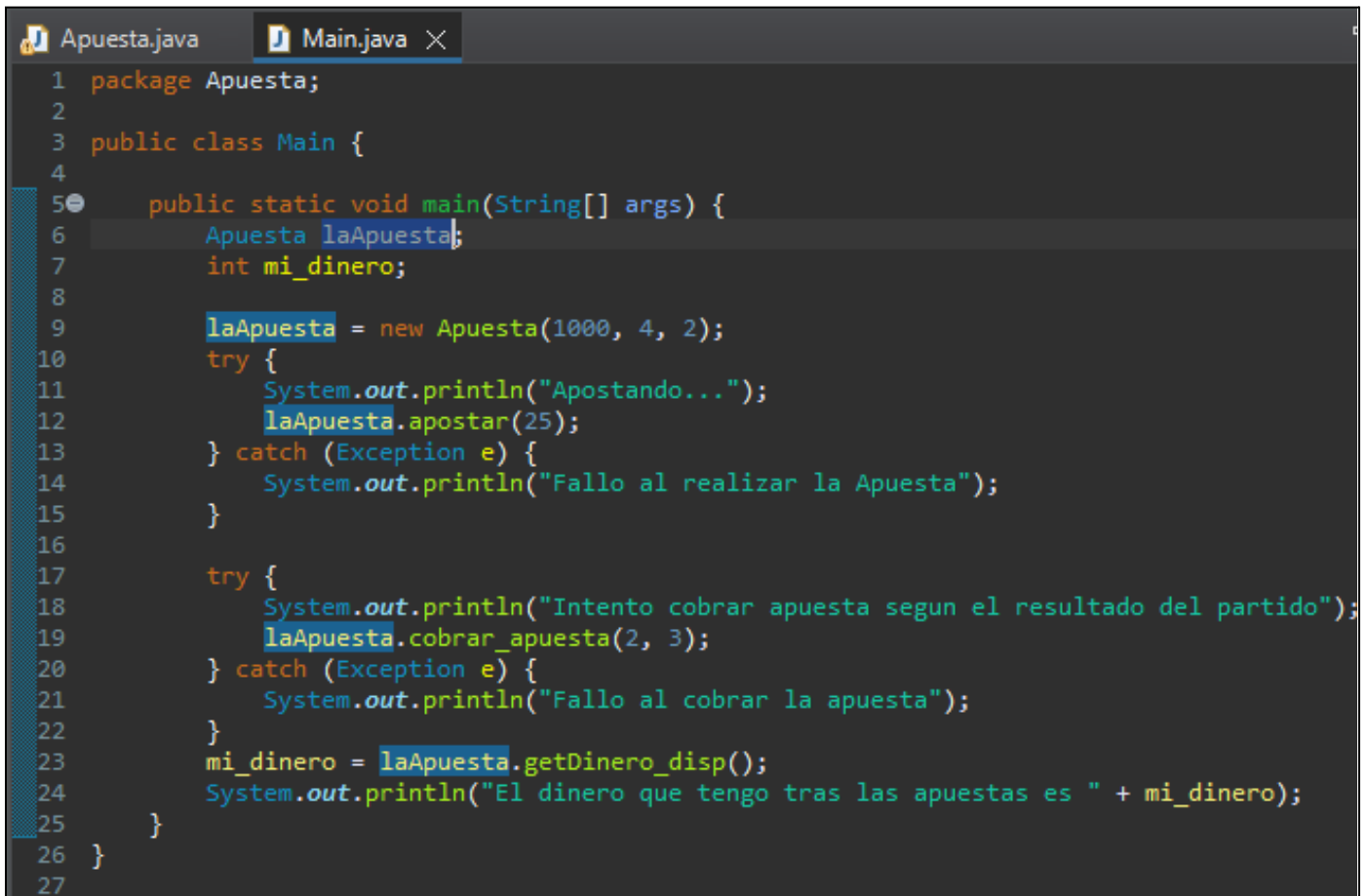
The screenshot shows the Eclipse IDE interface. The title bar reads "Eclipse - EntornoDesarrollo/src/Apuesta/Main.java - Eclipse IDE". The menu bar includes "File", "Edit", "Refactor", "Navigate", "Search", "Project", "Run", "Window", and "Help". The "Refactor" menu is open, and the "Rename" option is highlighted. The editor window shows the file "Main.java" with the following code:

```
1 package Apuesta;
2
3 public class Main {
4
5     public static void main(String[] args) {
6         Apuesta miApuesta;
7         int mi_dinero;
8
9         miApuesta = new Apuesta(1000, 4, 2);
10        try {
11            System.out.println("Apostando...");
12            miApuesta.apostar(25);
13        } catch (Exception e) {
14            System.out.println("Fallo al realizar la Apuesta");
15        }
16
17        try {
18            System.out.println("Intento cobrar apuesta segun el resultado d");
19            miApuesta.cobrar_apuesta(2, 3);
20        } catch (Exception e) {
21            System.out.println("Fallo al cobrar la apuesta");
22        }
23    }
24 }
```

The variable "miApuesta" is highlighted in blue on line 6, and the "Refactor" menu is open above it.



De esta forma, esté donde esté ubicada la variable, se renombrará con su nuevo nombre:



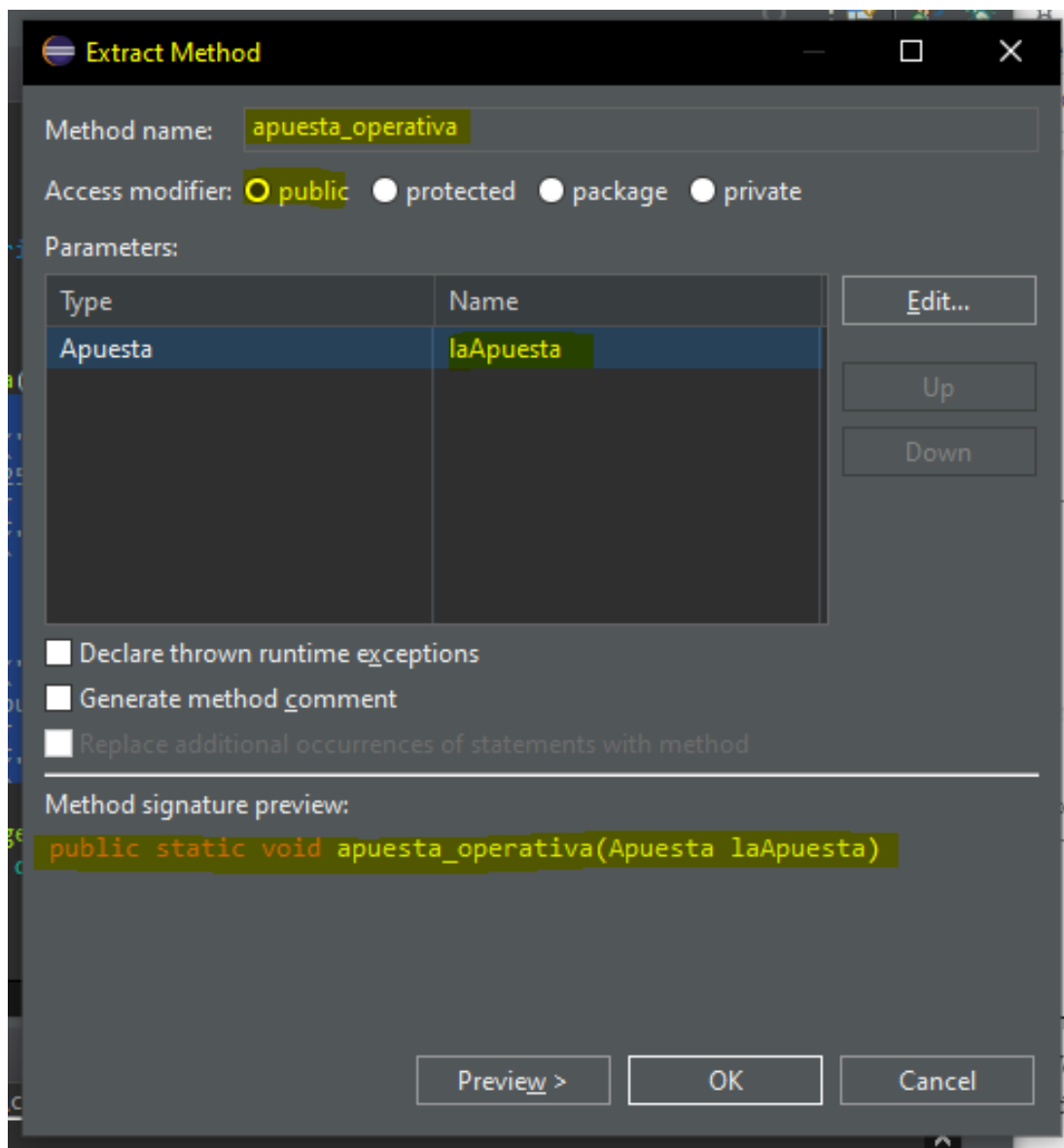
```
1 package Apuesta;
2
3 public class Main {
4
5     public static void main(String[] args) {
6         Apuesta laApuesta;
7         int mi_dinero;
8
9         laApuesta = new Apuesta(1000, 4, 2);
10        try {
11            System.out.println("Apostando...");
12            laApuesta.apostar(25);
13        } catch (Exception e) {
14            System.out.println("Fallo al realizar la Apuesta");
15        }
16
17        try {
18            System.out.println("Intento cobrar apuesta segun el resultado del partido");
19            laApuesta.cobrar_apuesta(2, 3);
20        } catch (Exception e) {
21            System.out.println("Fallo al cobrar la apuesta");
22        }
23        mi_dinero = laApuesta.getDinero_disp();
24        System.out.println("El dinero que tengo tras las apuestas es " + mi_dinero);
25    }
26 }
27
```

2.2. Método operativa Apuesta.

Introducimos el método “*apuesta_operativa*” en la clase “*main*”, ya que esta función engloba las sentencias que operan con el objeto “*laApuesta*” en la clase “*Main*” y que podrían estar “encapsuladas” en un método, ahorrando líneas de código.

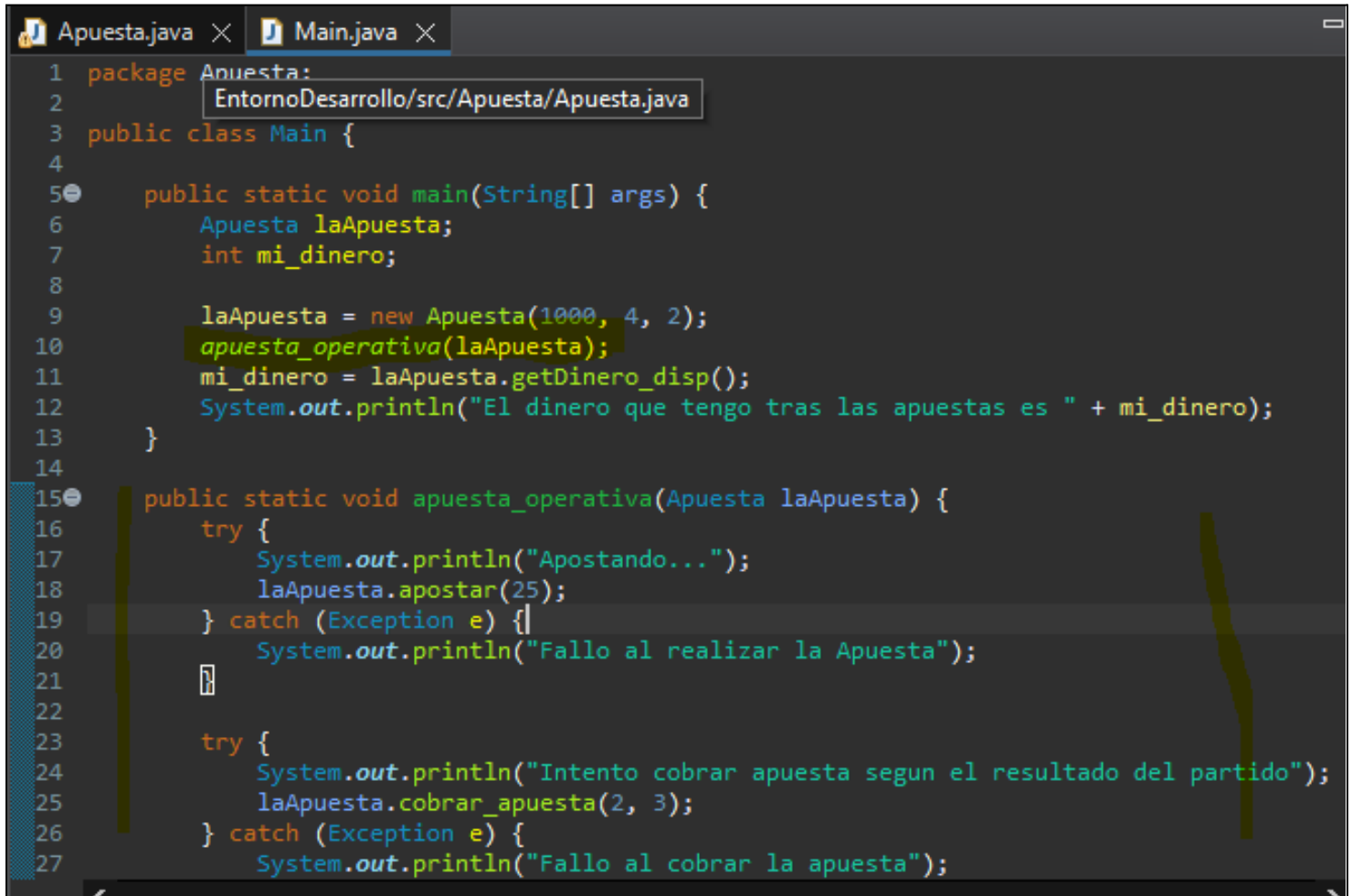
Para ello, seleccionamos dichas sentencias y volveremos a la opción “Refactor” < “*Extract Method*”:

```
1 package Apuesta;
2
3 public class Main {
4
5     public static void main(String[] args) {
6         Apuesta laApuesta;
7         int mi_dinero;
8
9         laApuesta = new Apuesta(1000, 4, 2);
10        try {
11            System.out.println("Apostando...");
12            laApuesta.apostar(25);
13        } catch (Exception e) {
14            System.out.println("Fallo al realizar la Apuesta");
15        }
16
17        try {
18            System.out.println("Intento cobrar apuesta segun el resultado del partido");
19            laApuesta.cobrar_apuesta(2, 3);
20        } catch (Exception e) {
21            System.out.println("Fallo al cobrar la apuesta");
22        }
23        mi_dinero = laApuesta.getDinero_disp();
24        System.out.println("El dinero que tengo tras las apuestas es " + mi_dinero);
25    }
26 }
27
```



Una vez en la ventana de “*Extract Method*”, llamamos a nuestro método “*apuesta_operativa*” y lo crearemos público. Vemos como en el campo de

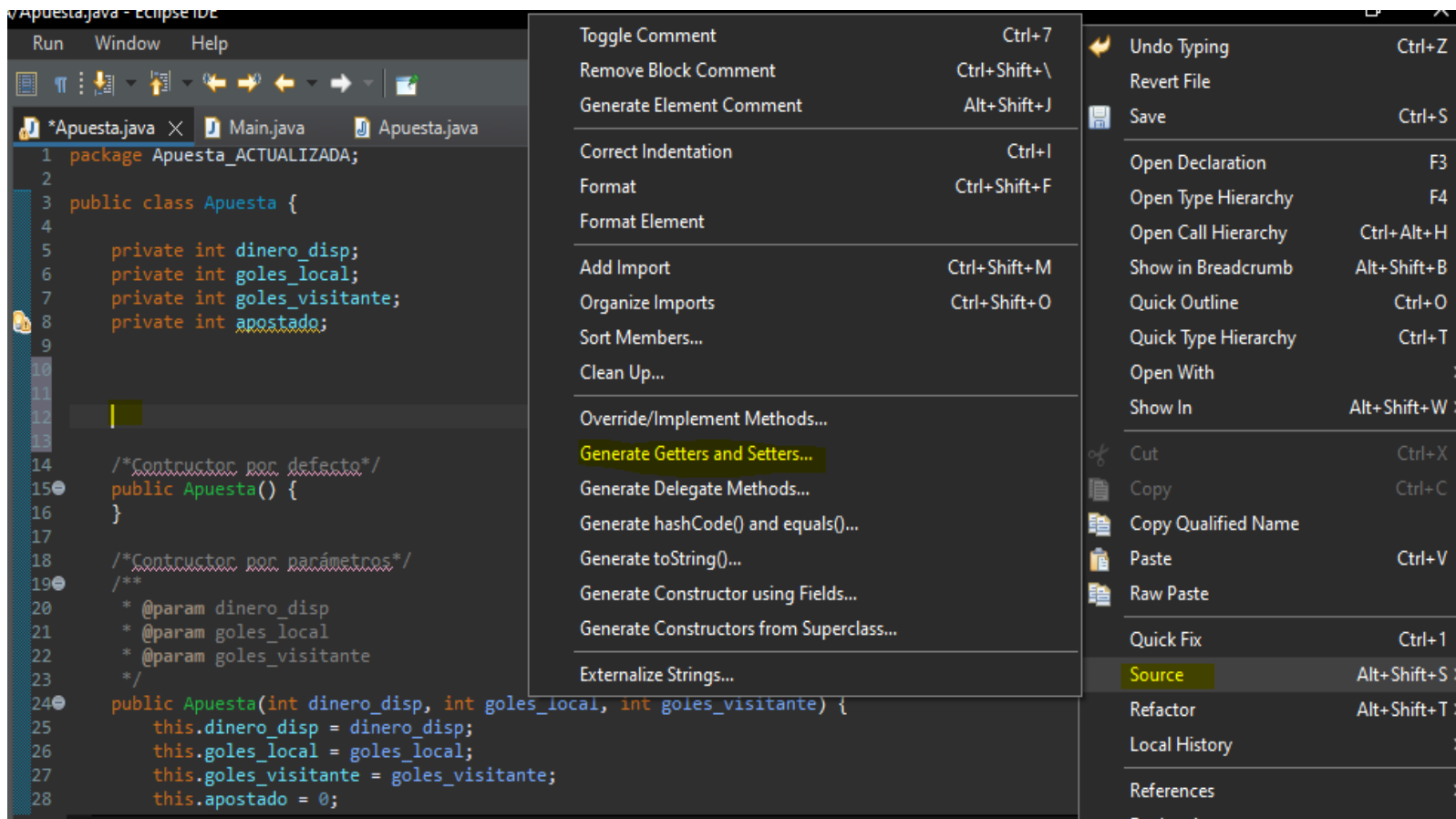
parámetros reconoce la instancia que se le pasará como parámetro de tipo “Apuesta” (clase). Al darle “ok”, automáticamente creará el método y la llamada en el lugar donde se encontraban estas sentencias:

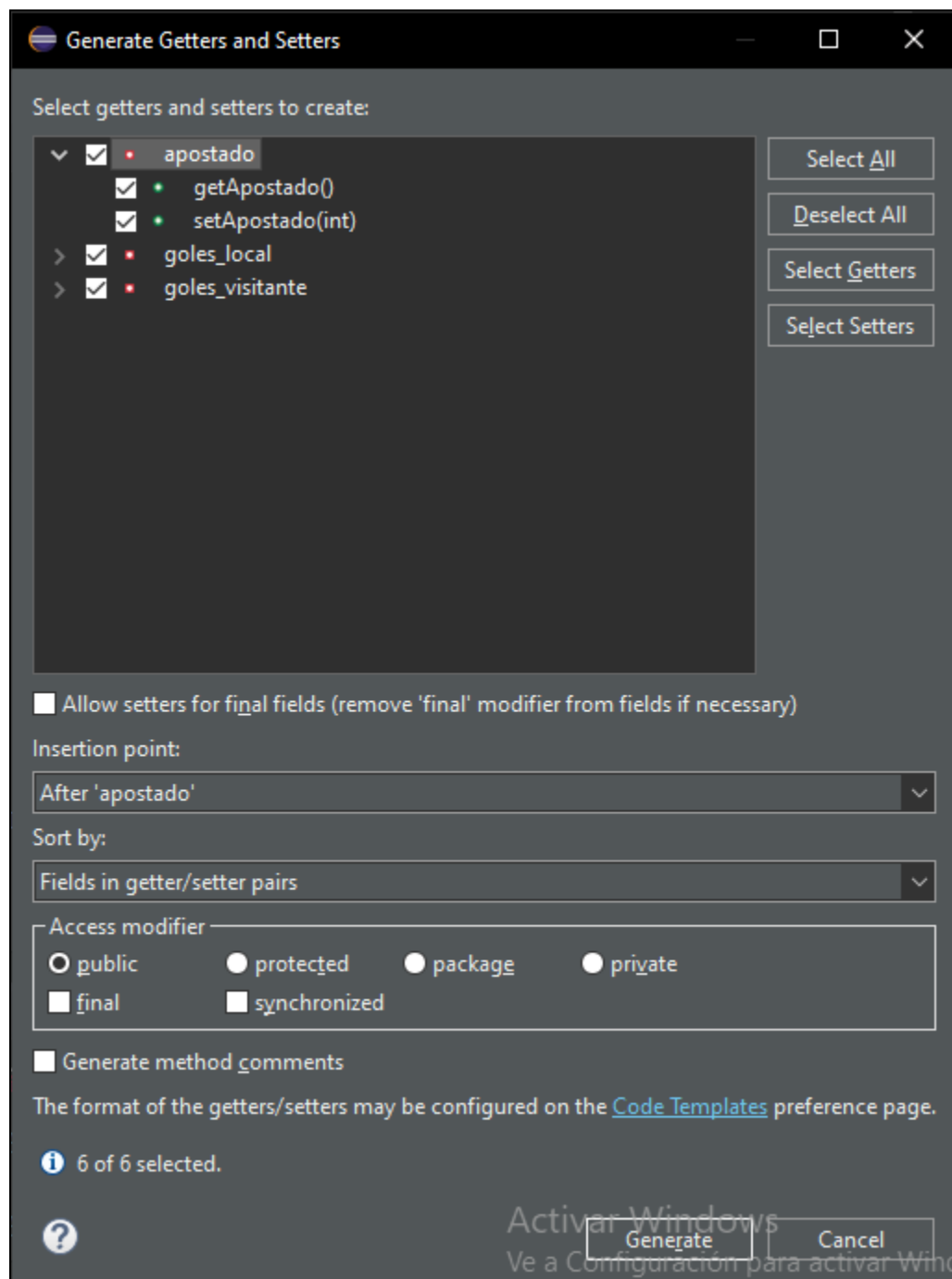


```
1 package Anuesta:
2   EntornoDesarrollo/src/Apuesta/Apuesta.java
3 public class Main {
4
5     public static void main(String[] args) {
6         Apuesta laApuesta;
7         int mi_dinero;
8
9         laApuesta = new Apuesta(1000, 4, 2);
10        apuesta_operativa(laApuesta);
11        mi_dinero = laApuesta.getDinero_disp();
12        System.out.println("El dinero que tengo tras las apuestas es " + mi_dinero);
13    }
14
15    public static void apuesta_operativa(Apuesta laApuesta) {
16        try {
17            System.out.println("Apostando...");
18            laApuesta.apostar(25);
19        } catch (Exception e) {}
20        System.out.println("Fallo al realizar la Apuesta");
21
22
23        try {
24            System.out.println("Intento cobrar apuesta segun el resultado del partido");
25            laApuesta.cobrar_apuesta(2, 3);
26        } catch (Exception e) {}
27        System.out.println("Fallo al cobrar la apuesta");
```

2.3. Encapsulamiento de los atributos de la clase Apuesta

Para encapsular los atributos de la clase “Apuesta”, nos dirigimos al código y clicamos botón derecho. Nos dirigimos a la opción “Source” < “Generate Getter and Setter”, elegimos los métodos *get* (nos da el valor del atributo) y *set* (nos ayuda insertar un valor en un atributo) que nos interesa de cada atributo, y automáticamente se generan:

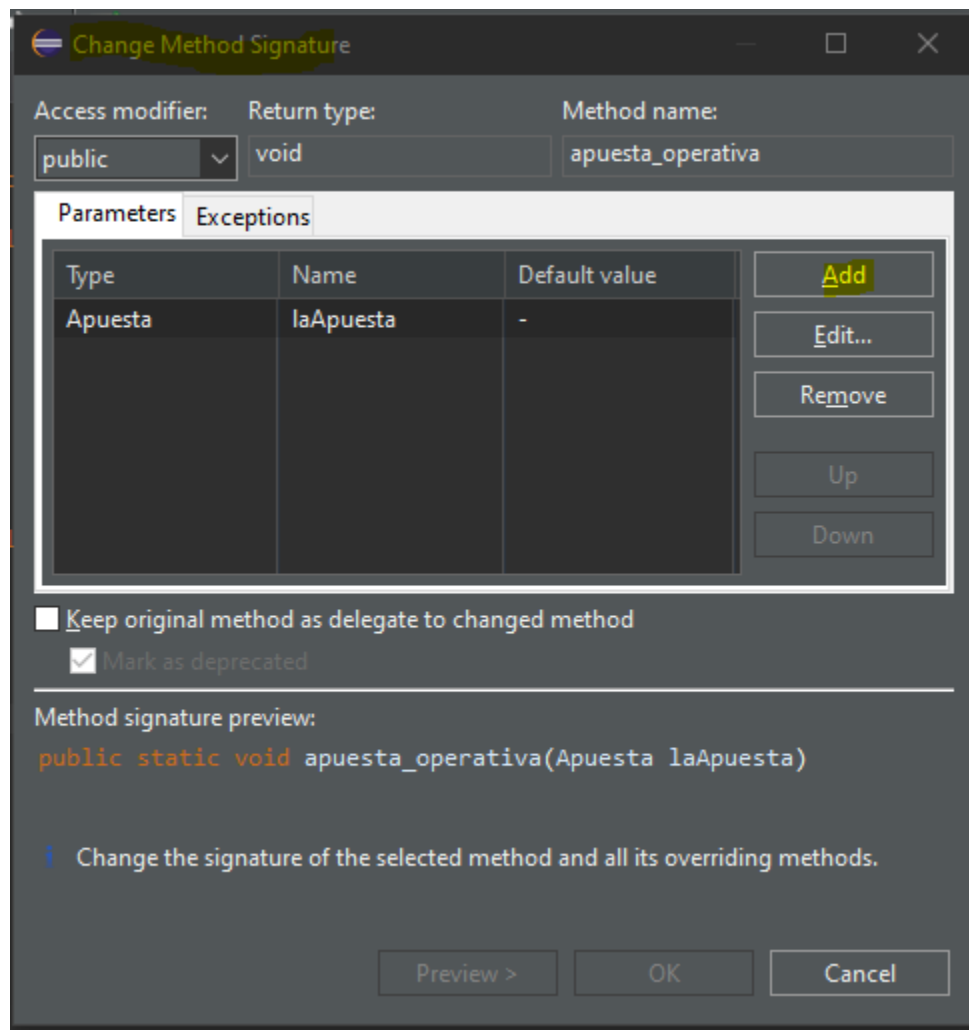




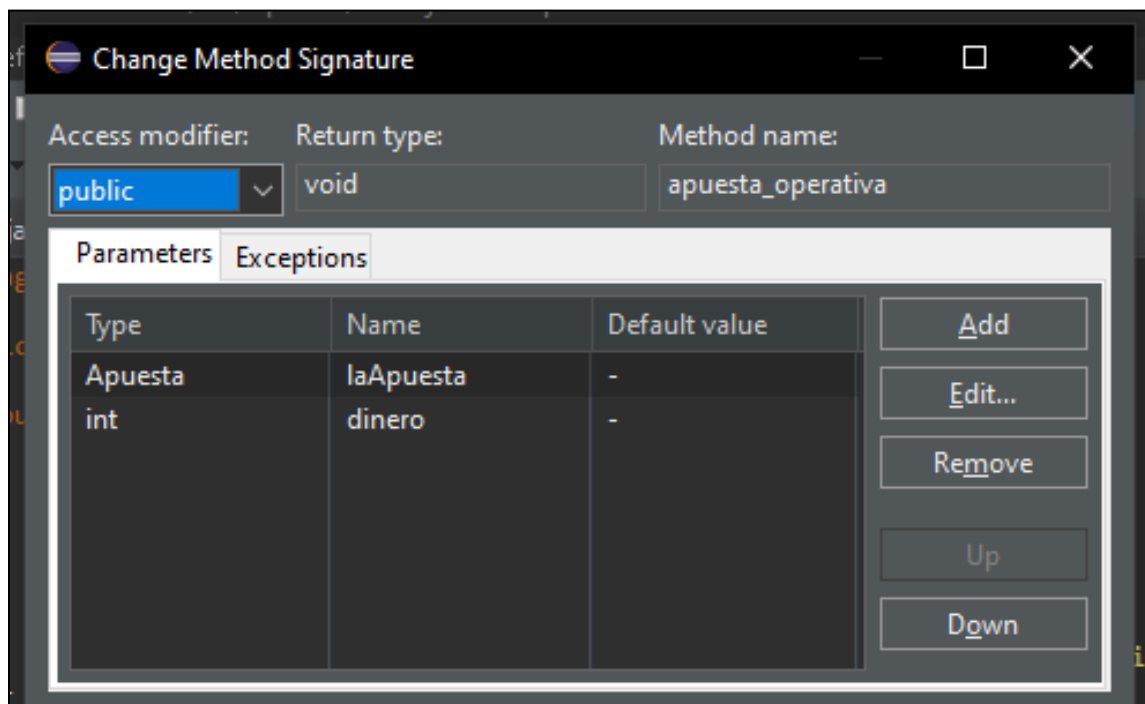
```
14 public int getGoles_local() {  
15     return goles_local;  
16 }  
17 public void setGoles_local(int goles_local) {  
18     this.goles_local = goles_local;  
19 }  
20 public int getGoles_visitante() {  
21     return goles_visitante;  
22 }  
23 public void setGoles_visitante(int goles_visitante) {  
24     this.goles_visitante = goles_visitante;  
25 }  
26 public int getApostado() {  
27     return apostado;  
28 }  
29 public void setApostado(int apostado) {  
30     this.apostado = apostado;  
31 }  
32 }
```

2. 4. Añadir un nuevo parámetro al método `operativa_Apuesta`, de nombre `dinero` y de tipo `int`.

Como veíamos anteriormente en el método “`operativa_Apuesta`”, sólo nos encontramos un parámetro de tipo objeto de la clase “`Apuesta`”. Ahora le añadiremos un nuevo parámetro de tipo `int` llamada “`dinero`”. Para ello, seleccionamos de nuevo la opción “*Refactor*” < “*Change Method Signature*”:



Añadimos dicho parámetro y :



```
Apuesta.java Main.java X
1 package Apuesta;
2
3 public class Main {
4
5     public static void main(String[] args) {
6         Apuesta laApuesta;
7         int mi_dinero;
8
9         laApuesta = new Apuesta(1000, 4, 2);
10        apuesta_operativa(laApuesta, 1000);
11        mi_dinero = laApuesta.getDinero_disp();
12        System.out.println("El dinero que tengo tras las apuestas es " + mi_dinero);
13    }
14
15    public static void apuesta_operativa(Apuesta laApuesta, int dinero) {
16        try {
17            System.out.println("Apostando...");
18            laApuesta.apostar(25);
19        } catch (Exception e) {
20            System.out.println("Falla al realizar la apuesta");
21        }
22    }
23 }
```

3. ANALIZADOR DE CÓDIGO

Para analizar el código, nos descargamos el plugin PMD, un analizador de código fuente multilenguaje. Según su página web (<https://pmd.github.io/>):

- *“PMD encuentra fallas de programación comunes como variables no utilizadas, bloques catch vacíos, creación de objetos innecesarios, etc. Es compatible con Java, JavaScript, Salesforce.com Apex y Visualforce, PLSQL, Apache Velocity, XML, XSL. Adicionalmente incluye CPD, el detector de copiar y pegar. CPD encuentra código duplicado en Java, C, C++, C#, Groovy, PHP, Ruby, Fortran, JavaScript, PLSQL, Apache Velocity, Scala, Objective C, Matlab, Python, Go, Swift y Salesforce.com Apex y Visualforce”.*

Procedemos a descargarnos PMD a través de la tienda del propio programa de Eclipse:

Eclipse Marketplace

Select solutions to install. Press Install Now to proceed with installation.
Press the "more info" link to learn more about a solution.



Search

Recent

Popular

Favorites

Installed

💡 Giving IoT an Edge

Find:



eclipse-pmd 3.5

The eclipse-pmd plug-in integrates the source code analyzer PMD into the Eclipse IDE. Everytime you save your work, eclipse-pmd scans your source code and looks... [more info](#)

by [Philip Graf](#), EPL 2.0
[PMD](#) [Static Code Analysis](#) [code analyzer](#) [code quality](#)

★ 307

 Installs: 198K (443 last month)



pmd-eclipse-plugin 4.44.0

PMD is a source code analyzer. It finds common programming flaws like unused variables, empty catch blocks, unnecessary object creation, and so forth. It supports... [more info](#)

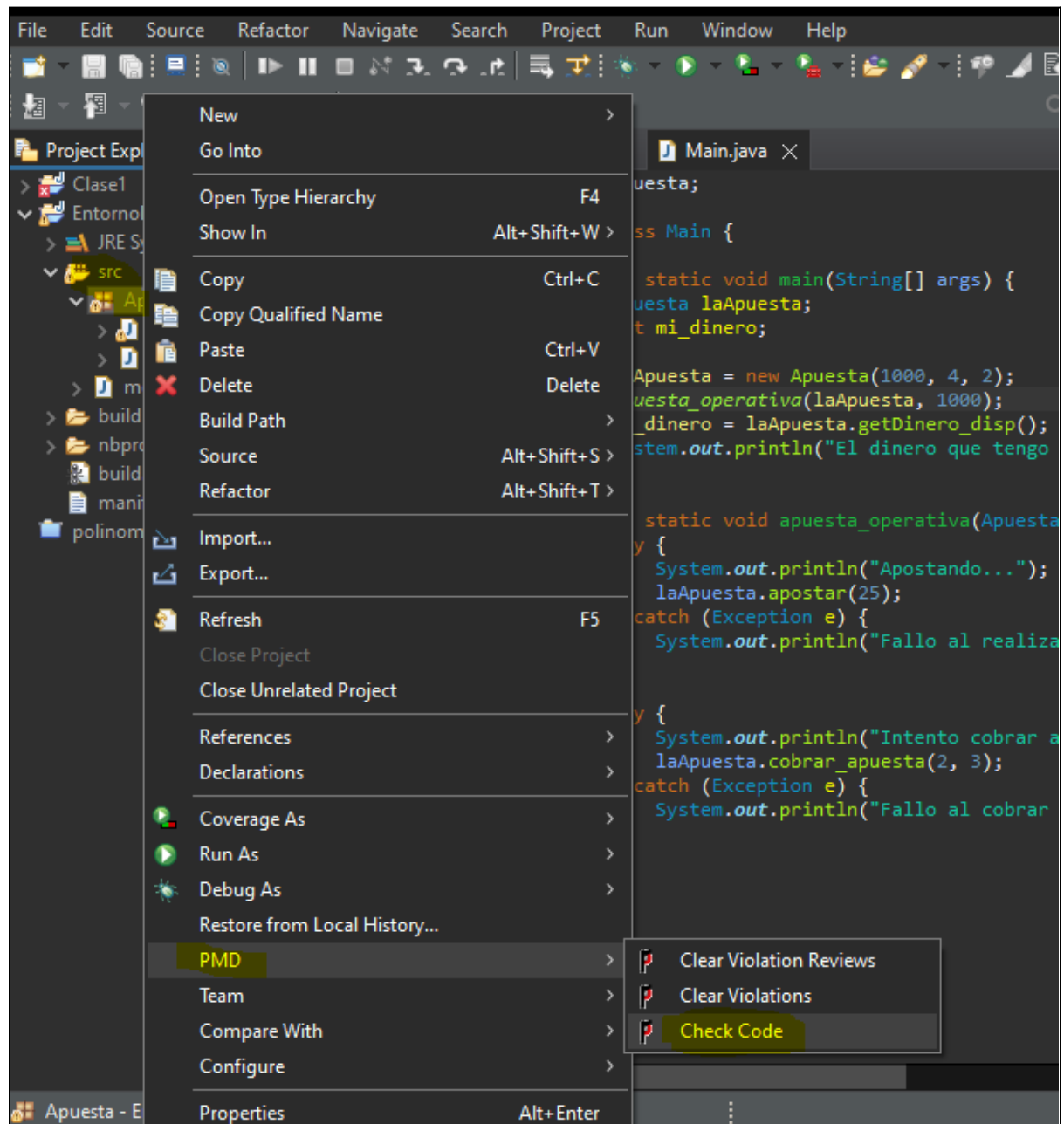
by [PMD](#), BSD
[PMD](#) [linter](#) [Source Code Analyzer](#) [code quality](#) [java](#)

★ 177

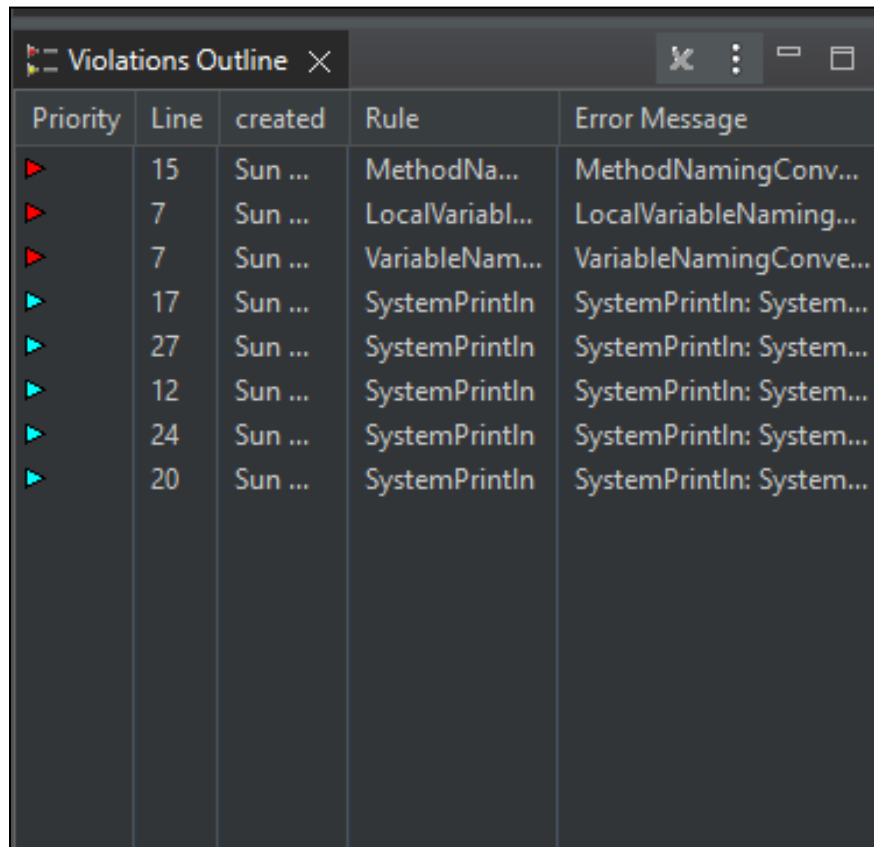
 Installs: 89,8K (1.237 last month)

3.1. Ejecución de PMD

Una vez descargado e instalado en Eclipse, seleccionamos con el botón derecho del ratón la carpeta donde se encuentre nuestro proyecto, y arrancamos 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 resultado es este:



Priority	Line	created	Rule	Error Message
▶	15	Sun ...	MethodNa...	MethodNamingConv...
▶	7	Sun ...	LocalVariabl...	LocalVariableNaming...
▶	7	Sun ...	VariableNam...	VariableNamingConve...
▶	17	Sun ...	SystemPrintln	SystemPrintln: System...
▶	27	Sun ...	SystemPrintln	SystemPrintln: System...
▶	12	Sun ...	SystemPrintln	SystemPrintln: System...
▶	24	Sun ...	SystemPrintln	SystemPrintln: System...
▶	20	Sun ...	SystemPrintln	SystemPrintln: System...

Violations Overview					
Element	# Violations	# Violations/KLOC	/violations/Method	Project	
▼ Apuesta	34	653.8	3.78	EntornoDesarro...	
▼ Main.java	8	444.4	4.00	EntornoDesarro...	
▶ VariableNamingConventions	1	55.6	0.50	EntornoDesarro...	
▶ LocalVariableNamingConventions	1	55.6	0.50	EntornoDesarro...	
▶ MethodNamingConventions	1	55.6	0.50	EntornoDesarro...	
▶ SystemPrintln	5	277.8	2.50	EntornoDesarro...	
▼ Apuesta.java	26	764.7	3.71	EntornoDesarro...	
▶ FormalParameterNamingConventio	6	176.5	0.86	EntornoDesarro...	
▶ AvoidThrowingRawExceptionTypes	4	117.6	0.57	EntornoDesarro...	
▶ VariableNamingConventions	9	264.7	1.29	EntornoDesarro...	
▶ MethodNamingConventions	4	117.6	0.57	EntornoDesarro...	
▶ FieldNamingConventions	3	88.2	0.43	EntornoDesarro...	

La interfaz PMD, nos permite reconocer los problemas de cada clase y en la línea donde se ha detectado el error, incluso con una pequeña descripción si dejamos el ratón por encima del símbolo marcado en el código:

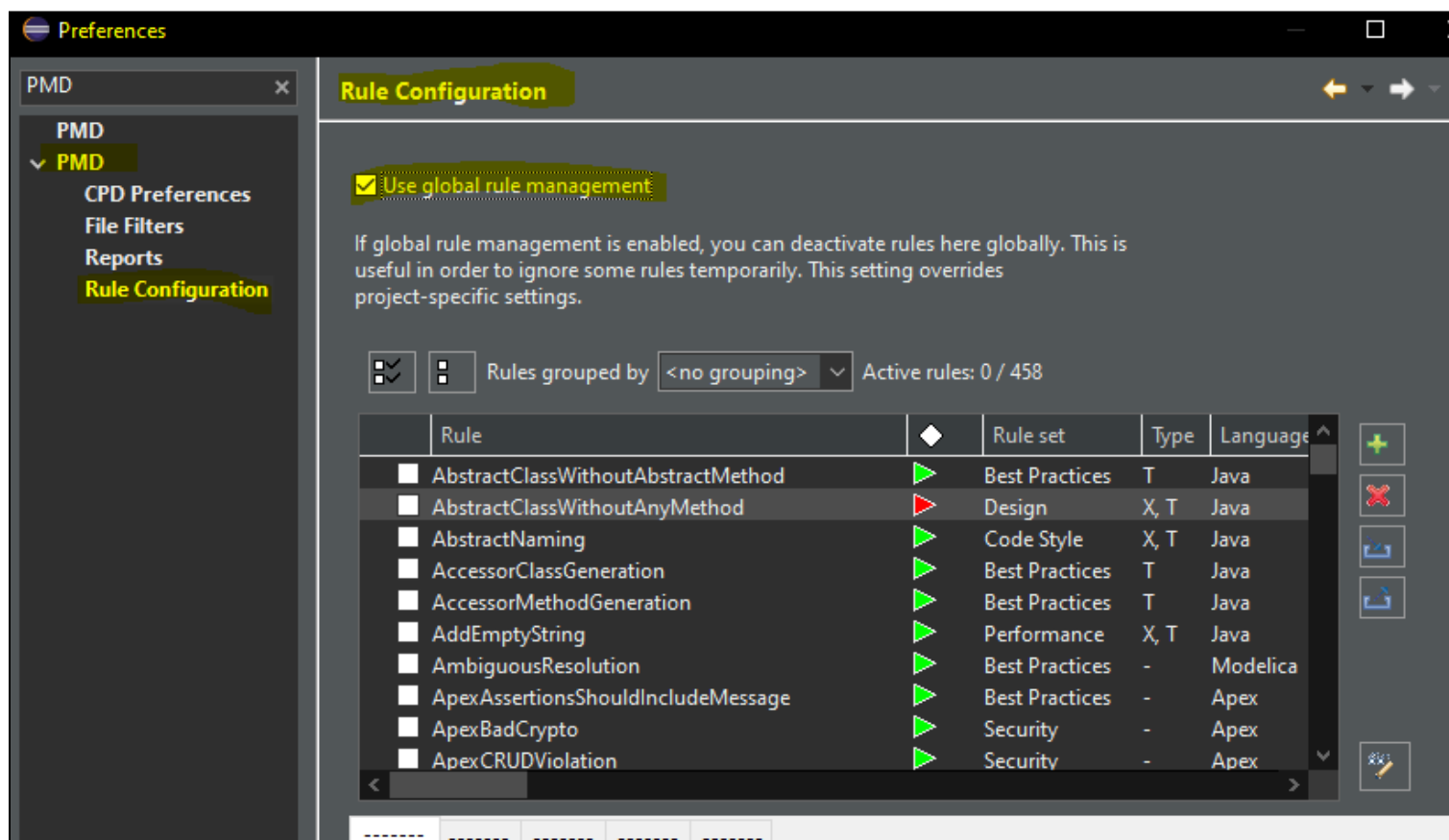
```

Apuesta.java Main.java
1 package Apuesta;
2
3 public class Main {
4
5     public static void main(String[] args) {
6         Apuesta laApuesta;
7         ▶ Multiple markers at this line
8           - VariableNamingConventions: Only variables that are final should contain underscores (except for underscores in standard prefix/suffix), 'mi_dinero' is not final.
9           - LocalVariableNamingConventions: The local variable name 'mi_dinero' doesn't match '[a-z][a-zA-Z0-9]*'
10        apuesta_operativa(laApuesta, 1000);
11        mi_dinero = laApuesta.getDinero_disp();
12        System.out.println("El dinero que tengo tras las apuestas es " + mi_dinero);
13    }

```

3.2. Configuración del analizador de código PMD: escaneado automático del proyecto, a intervalos regulares y adicción de tres nuevas reglas

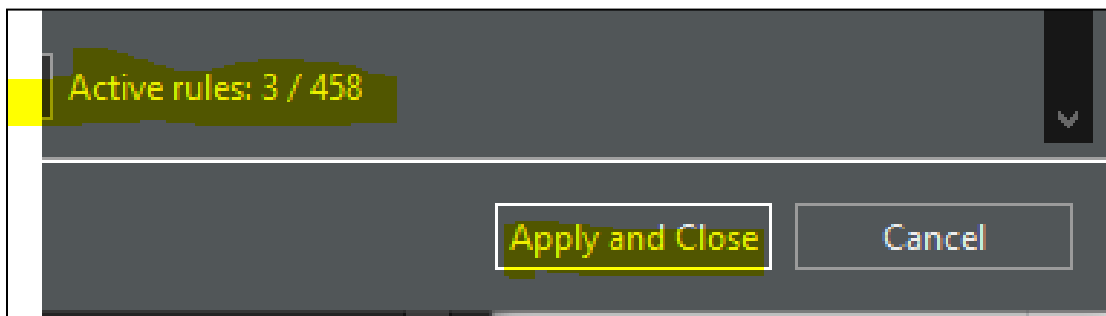
Para navegar entre las opciones y preferencias de PMD, tendremos que irnos a la opción de “*window*” < “*preferences*”, al apartado de PMD. Para añadir tres nuevas reglas de chequeo sólo tendremos que activar la opción “*global rule management*” en el apartado de “*Rule Configuration*”, y añadimos las reglas que más nos convengan:



En esta ocasión, elegiremos tres nuevas reglas:

- *AssignmentToNonFinalStatic* → Identifica un uso inseguro de un campo estático.
- *FinalParameterInAbstractMethod* → Detecta parámetros declarados como final en métodos de interfaz, ya que éstos pueden ser no respetados en la implementación.
- *OnlyOneReturn* → Un método debe de tener sólo un punto de salida, y esa debe de ser la última instrucción del método.

Aplicamos estos cambios, y cerramos el menú de “*Preferencias*”

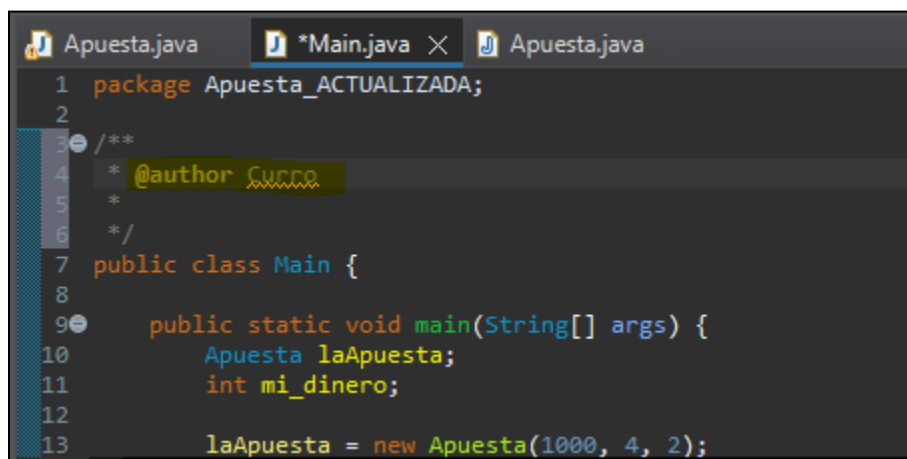
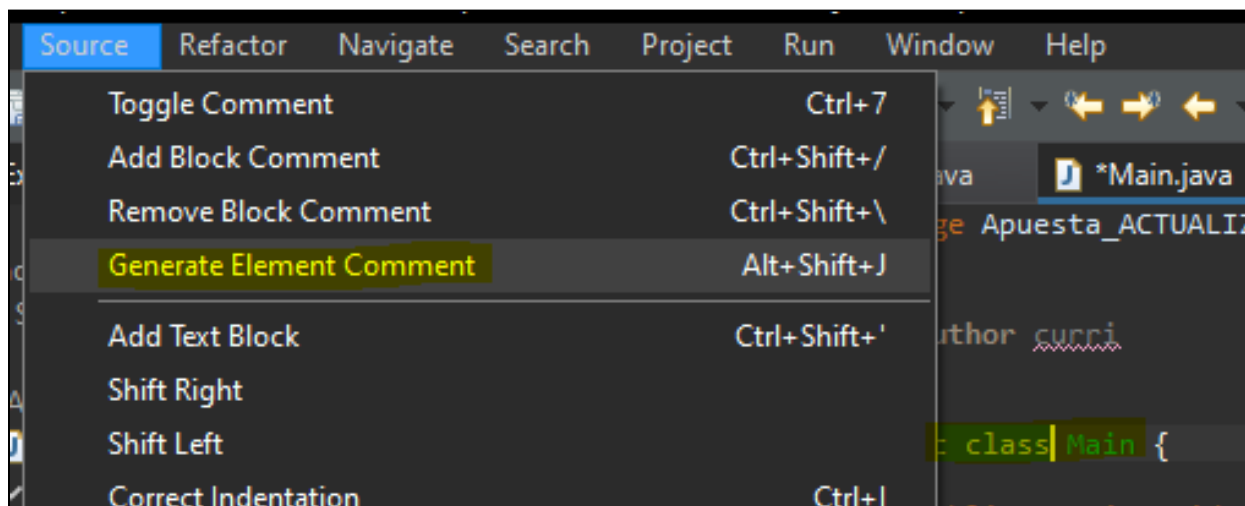


NOTA: El escaneado automático por intervalos no será posible realizarlo ya que PMD Eclipse no da esa funcionalidad.

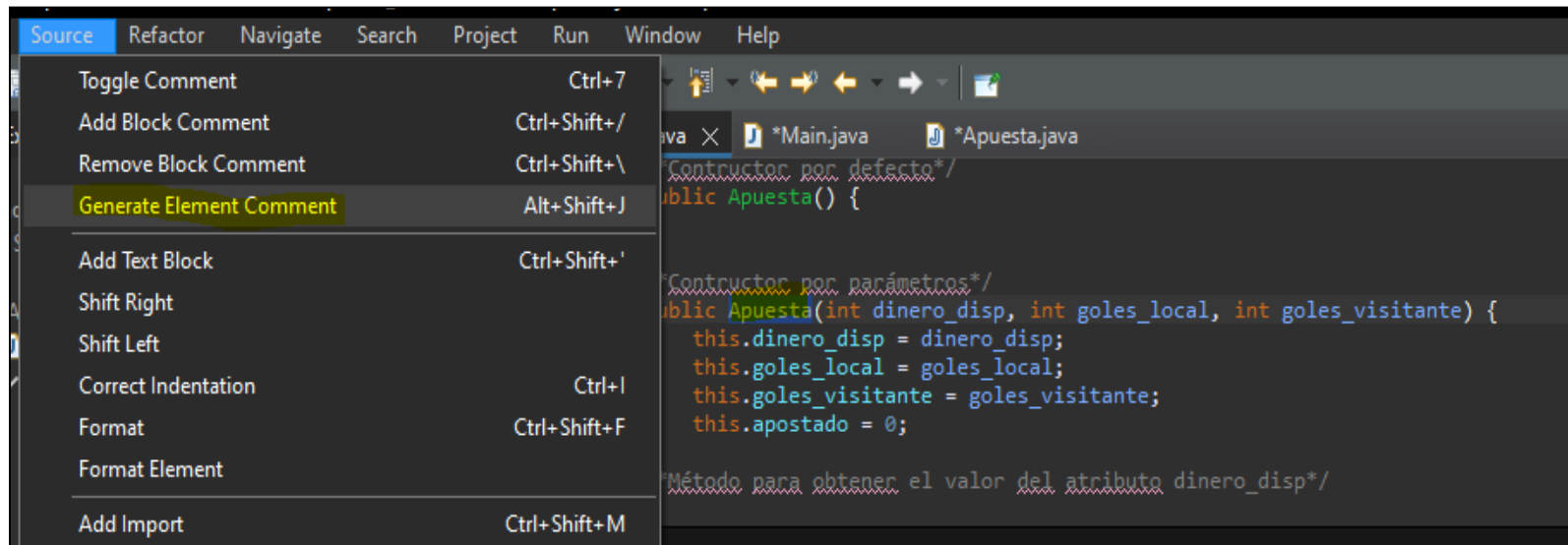
4. JAVADOC

4.1. Inserción de comentarios para el JavaDoc.

Para insertar comentarios en formato JavaDoc, tenemos que seleccionar los elementos del código, como una clase o un método, e irnos a la opción “Source” < “Generate Element Comment”. En este ejemplo hemos seleccionado la clase Main y se nos ha generado la etiqueta de autor:



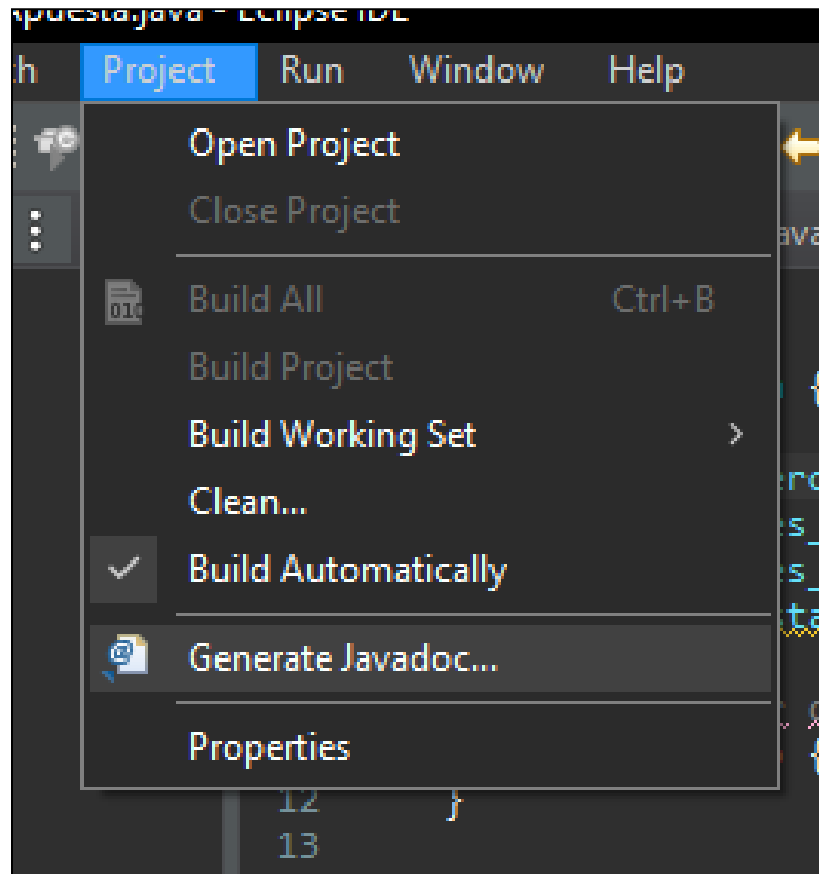
Lo mismo podemos hacer para comentar un método (de nombre Apuesta), de la clase Apuesta:



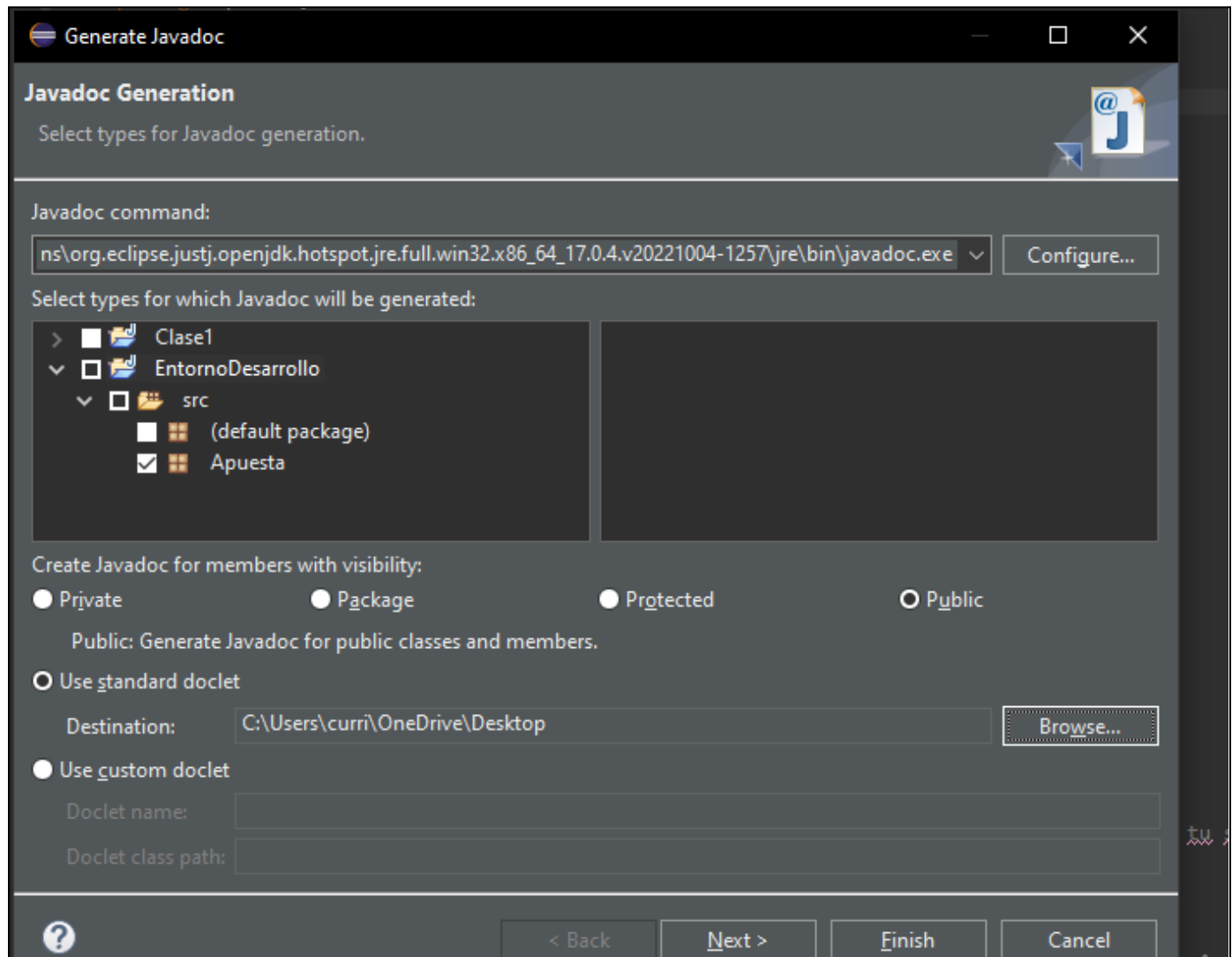
```
14  /*Constructor por parámetros*/
15  /**
16   * @param dinero_disp
17   * @param goles_local
18   * @param goles_visitante
19   */
20  public Apuesta(int dinero_disp, int goles_local, int goles_visitante) {
21      this.dinero_disp = dinero_disp;
22      this.goles_local = goles_local;
```

4.2. Generación de documentación Javadoc del proyecto Apuesta en Eclipse.

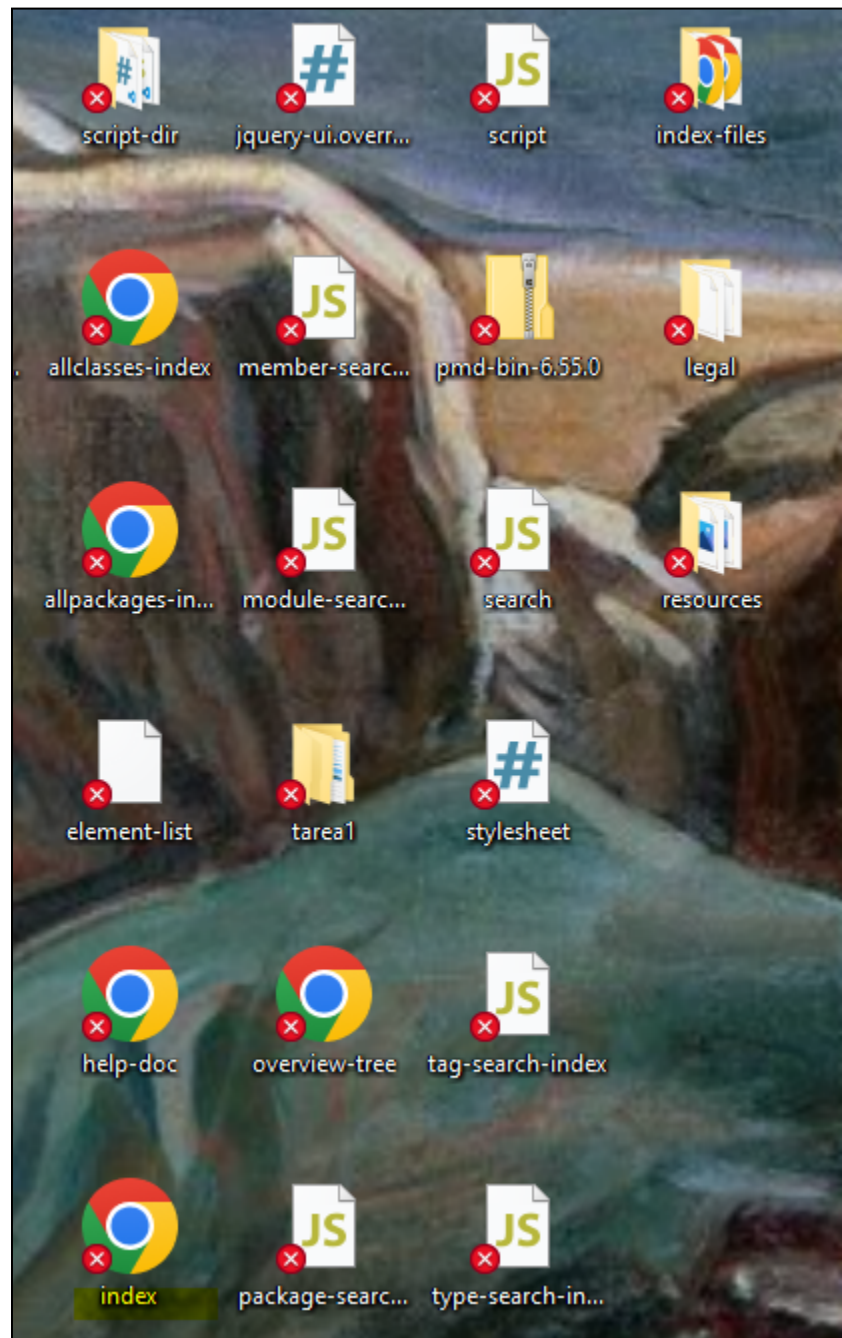
Para generar la documentación Javadoc de todo nuestro proyecto tendremos que clicar en la opción “*Project*” < “*Generate Javadoc*”:



Se nos abre una nueva ventana donde tendremos que seleccionar nuestro proyecto “Apuesta” -ya actualizada- y el destino de los futuros archivos generados de Javadoc:



Automáticamente se nos generará los siguientes archivos. Pinchamos en el HTML generado con nombre “INDEX”.



MODULE PACKAGE **CLASS** USE TREE INDEX HELP

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

Module EntornoDesarrollo
Package Apuesta_ACTUALIZADA
Class Main
java.lang.Object
Apuesta_ACTUALIZADA.Main

```
public class Main  
extends Object
```

Author:
Curro

Constructor Summary

Constructors	
Constructor	Description
Main()	

Method Summary

All Methods	Static Methods	Concrete Methods
-------------	----------------	------------------

De esta forma, obtenemos nuestro proyecto comentado automáticamente en formato HTML. Como vemos en la imagen de arriba, vemos una de las etiquetas -“*Author*”- que añadimos anteriormente.

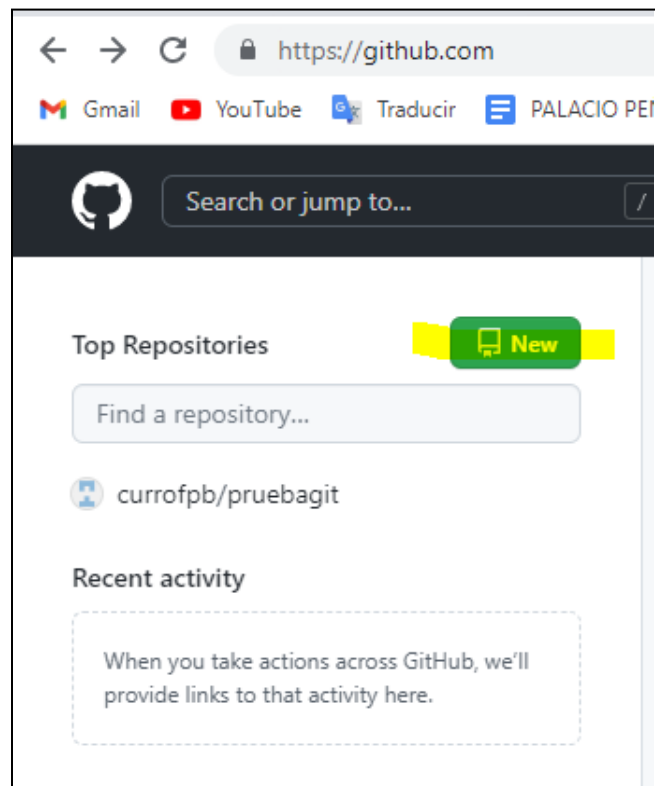
5. GitHub y entrega del documento

GitHub es una plataforma online de repositorios GIT que actúa como un controlador de versiones.

Para la entrega de este proyecto, utilizaremos en su entrega un repositorio de GitHub, subiendo en una rama este pdf y la documentación JavaDoc, y por otra, el código original del paquete Apuesta (clases Apuesta y Main) y el mismo paquete actualizado (refactorización).

5.1. Creación de Repositorio Público en GitHub.com

Iniciamos nuestra cuenta GitHub -ya creada en anteriores tareas- y clicamos el icono del libro “New” para crear un nuevo repositorio:




En la nueva página, estableceremos las características de nuestro repositorio. Introduciremos su nombre, lo haremos público para que usuarios externos puedan verlo y añadiremos una descripción (opcional):

Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository.](#)

Owner *

Repository name *

 currofpb ▾


/

OptimizacionDocumentacion ✓


Great repository names are short and memorable. Need inspiration? How about [silver-invention?](#)

Description (optional)

Repositorio para depositar los archivos referentes a la tarea del tema 4 de Entornos de Desarrollo, Optimizació

☒  **Public**

Anyone on the internet can see this repository. You choose who can commit.

☐  **Private**

You choose who can see and commit to this repository.

Initialize this repository with:

Skip this step if you're importing an existing repository.

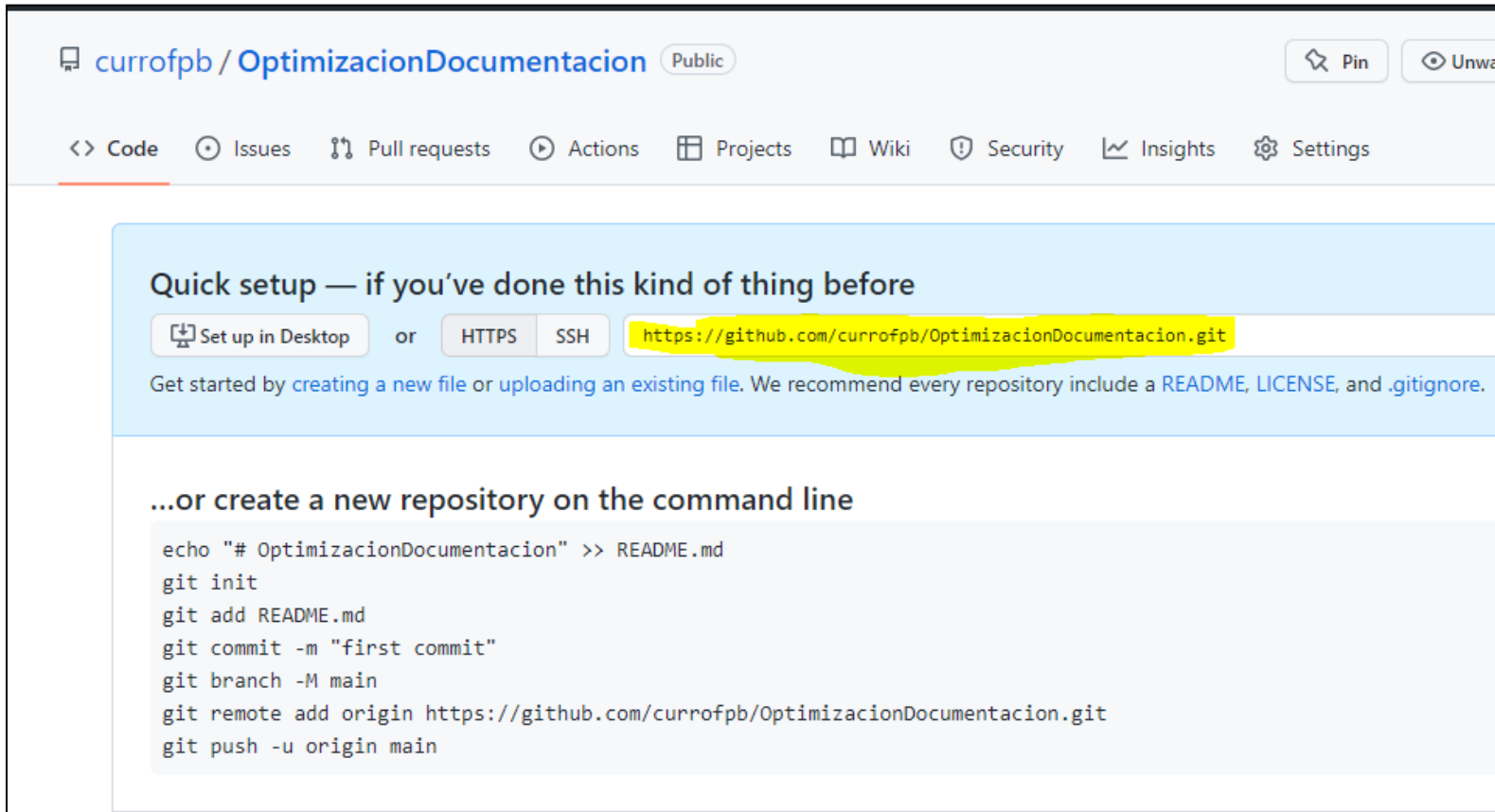
☐ **Add a README file**

This is where you can write a long description for your project. [Learn more.](#)

 You are creating a public repository in your personal account.

Create repository

Una vez clicamos “*Create repository*”, nos quedaremos con la *url* que nos servirá de enlace a la hora de subir los archivos:



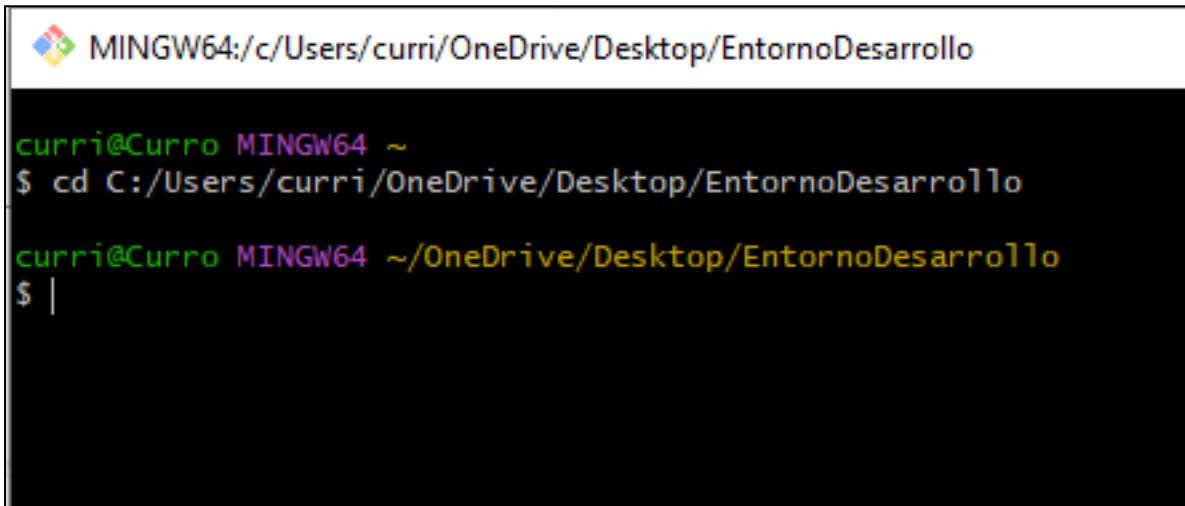
URL: <https://github.com/currofpb/OptimizacionDocumentacion.git>

5.2. Conexión GIT con nuestro repositorio currofpb/OptimizaciónDocumentacion.

Abrimos nuestra consola GIT -previamente instalada- y, a través de

comandos, nos conectaremos a nuestro repositorio creado anteriormente. Para ello, tenemos que ubicarnos en la carpeta la cual vamos a subir en el repositorio, con el comando “*cd*” seguido de la ruta de nuestra carpeta:

- “*cd C:/Users/curri/OneDrive/Desktop/EntornoDesarrollo*”.

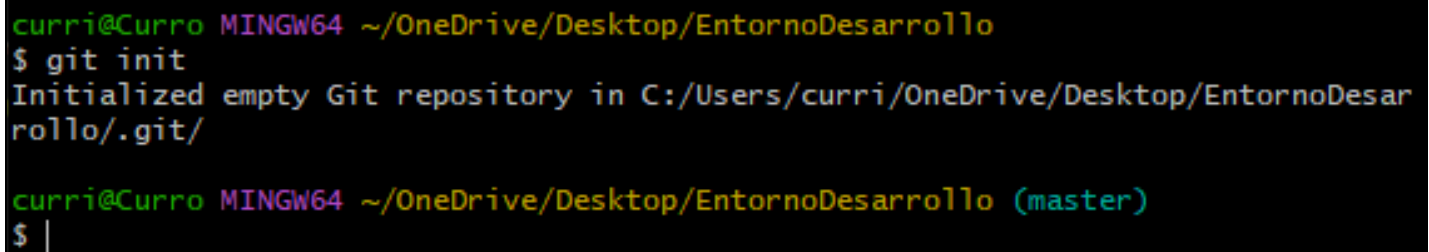


```
MINGW64:/c/Users/curri/OneDrive/Desktop/EntornoDesarrollo

curri@Curro MINGW64 ~
$ cd C:/Users/curri/OneDrive/Desktop/EntornoDesarrollo

curri@Curro MINGW64 ~/OneDrive/Desktop/EntornoDesarrollo
$ |
```

Una vez ubicados, tenemos que verificar si podemos iniciar GIT con el comando “*git init*”, inicializando un repositorio local vacío en la ubicación:



```
curri@Curro MINGW64 ~/OneDrive/Desktop/EntornoDesarrollo
$ git init
Initialized empty Git repository in C:/Users/curri/OneDrive/Desktop/EntornoDesarrollo/.git/

curri@Curro MINGW64 ~/OneDrive/Desktop/EntornoDesarrollo (master)
$ |
```

Por último, tenemos que identificarnos en este repositorio con nuestro nombre y correo con los comando “*git config name.(name//email)*”:

```
curri@Curro MINGW64 ~/OneDrive/Desktop/EntornoDesarrollo (master)
$ git config user.name "currofpb"

curri@Curro MINGW64 ~/OneDrive/Desktop/EntornoDesarrollo (master)
$ git config user.email "fj.fernandezprobarbarroja@iesrodmarin.com"
```

5.2. Conectar repositorio local con repositorio remoto.

Para comunicar el repositorio local con el remoto, tenemos que introducir el siguiente comando junto un nombre-contenedor (origin) para la url que nos proporcionó GitHub cuando creamos nuestro repositorio “OptimizaciónDocumentación”:

- *git remote add origin*
“*https://github.com/currofpb/OptimizacionDocumentacion.git*”.

```
curri@Curro MINGW64 ~/OneDrive/Desktop/EntornoDesarrollo (master)
$ git remote add origin "https://github.com/currofpb/OptimizacionDocumentacion.git"

curri@Curro MINGW64 ~/OneDrive/Desktop/EntornoDesarrollo (master)
$ |
```


5.3. Agregar primera rama al repositorio remoto desde nuestro repositorio local: archivos java Apuesta (código original y actualizado).

Para subir los archivos referentes al código de nuestro proyecto al repositorio remoto, tendremos que ejecutar el comando “add”. Pero primero, vemos el estado de nuestra carpeta con el comando “status”:

```
curri@Curro MINGW64 ~/OneDrive/Desktop/EntornoDesarrollo (master)
$ git status
On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
  Apuesta/
  Apuesta_Actualizada/
  JavaDoc_Apuesta/
  module-summary.html

nothing added to commit but untracked files present (use "git add" to track)
```

Como vemos, los archivos de la carpeta están en color rojo ya que aún no están subidos. Ahora sí, añadiremos de forma temporal con el comando “git add” más el nombre del archivo, y confirmaremos con el comando “commit -m” para que se guarden los cambios permanentes:

- “git add Apuesta/”
- “git add Apuesta_Actualizada”

```
curri@Curro MINGW64 ~/OneDrive/Desktop/EntornoDesarrollo (master)
$ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
    new file:   Apuesta/Apuesta.html
    new file:   Apuesta/Main.html
    new file:   Apuesta/class-use/Apuesta.html
    new file:   Apuesta/class-use/Main.html
    new file:   Apuesta/package-summary.html
    new file:   Apuesta/package-tree.html
    new file:   Apuesta/package-use.html
    new file:   Apuesta_Actualizada/Apuesta.html
    new file:   Apuesta_Actualizada/Apuesta.java
    new file:   Apuesta_Actualizada/Main.html
    new file:   Apuesta_Actualizada/Main.java
    new file:   Apuesta_Actualizada/class-use/Apuesta.html
    new file:   Apuesta_Actualizada/class-use/Main.html
    new file:   Apuesta_Actualizada/module-summary.html
    new file:   Apuesta_Actualizada/package-summary.html
    new file:   Apuesta_Actualizada/package-tree.html
    new file:   Apuesta_Actualizada/package-use.html

Untracked files:
  (use "git add <file>..." to include in what will be committed)
    JavaDoc_Apuesta/
    module-summary.html
```

- “Git commit -m “commit 1 códigos”.

```
curri@Curro MINGW64 ~/OneDrive/Desktop/EntornoDesarrollo (master)
$ git commit -m "Commit 1 códigos"
[master (root-commit) df70fcd] Commit 1 códigos
17 files changed, 1680 insertions(+)
create mode 100644 Apuesta/Apuesta.html
create mode 100644 Apuesta/Main.html
create mode 100644 Apuesta/class-use/Apuesta.html
create mode 100644 Apuesta/class-use/Main.html
create mode 100644 Apuesta/package-summary.html
create mode 100644 Apuesta/package-tree.html
create mode 100644 Apuesta/package-use.html
create mode 100644 Apuesta_Actualizada/Apuesta.html
create mode 100644 Apuesta_Actualizada/Apuesta.java
create mode 100644 Apuesta_Actualizada/Main.html
create mode 100644 Apuesta_Actualizada/Main.java
create mode 100644 Apuesta_Actualizada/class-use/Apuesta.html
create mode 100644 Apuesta_Actualizada/class-use/Main.html
create mode 100644 Apuesta_Actualizada/module-summary.html
create mode 100644 Apuesta_Actualizada/package-summary.html
create mode 100644 Apuesta_Actualizada/package-tree.html
create mode 100644 Apuesta_Actualizada/package-use.html
```

Ahora, con el comando “git push -u” subiremos los archivos añadidos en nuestro repositorio local. Para ello, añadiremos a este comando el nombre-contenedor de nuestro repositorio remoto (origin) y la rama en la que estamos trabajando (master, la principal):

- “git push -u origin master”.

```
curri@Curro MINGW64 ~/OneDrive/Desktop/EntornoDesarrollo (master)
$ git push -u origin master
Enumerating objects: 23, done.
Counting objects: 100% (23/23), done.
Delta compression using up to 8 threads
Compressing objects: 100% (23/23), done.
Writing objects: 100% (23/23), 8.72 KiB | 2.91 MiB/s, done.
Total 23 (delta 13), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (13/13), done.
To https://github.com/currofpb/OptimizacionDocumentacion.git
 * [new branch]      master -> master
branch 'master' set up to track 'origin/master'.
```

Comprobamos que se han subido dichos archivos a nuestro repositorio remoto:

The screenshot shows the GitHub interface for the repository 'currofpb / OptimizacionDocumentacion'. The repository is public and has 1 branch (master) and 0 tags. The commit history shows a single commit by 'currofpb' 12 minutes ago, with a commit hash of 'df70fcd'. The commit message is 'Commit 1 códigos'. The files listed in the commit are 'Apuesta' and 'Apuesta_Actualizada', both committed 12 minutes ago. The repository name 'OptimizacionDocumentacion' is highlighted in yellow. The commit message 'Commit 1 códigos' and the file names 'Apuesta' and 'Apuesta_Actualizada' are also highlighted in yellow.

File	Commit	Time
Apuesta	Commit 1 códigos	12 minutes ago
Apuesta_Actualizada	Commit 1 códigos	12 minutes ago

5.4. Creación de una segunda rama para subir los archivos pdf y documentación Javadoc.

Para crear una nueva rama o espacio de trabajo, tendremos, primero, que saber en qué rama estamos situados. Por ahora, sólo tenemos una rama principal llamada Master, pero aún así, verificamos nuestra ubicación con el comando “*git branch*”:

```
curri@Curro MINGW64 ~/OneDrive/Desktop/EntornoDesarrollo (master)
$ git branch
* master
```

Ahora crearemos otra rama llamada “rama_texto” -en referencia a los archivos de carácter número alfabético- con el mismo comando añadiendo el nombre:

- “*git branch rama_texto*”.

```
curri@Curro MINGW64 ~/OneDrive/Desktop/EntornoDesarrollo (master)
$ git branch rama_texto

curri@Curro MINGW64 ~/OneDrive/Desktop/EntornoDesarrollo (master)
$ git branch
* master
  rama_texto
```

Por último, tendremos que cambiar de rama con el comando “*git checkout rama_texto*”. Vemos como el id de la rama en la terminal GIT ha cambiado a “rama_texto”:

```
curri@Curro MINGW64 ~/OneDrive/Desktop/EntornoDesarrollo (master)
$ git checkout rama_texto
Switched to branch 'rama_texto'

curri@Curro MINGW64 ~/OneDrive/Desktop/EntornoDesarrollo (rama_texto)
$ |
```

Una vez en este punto, sólo tendremos que repetir el mismo proceso anterior para añadir y subir los archivos restantes del trabajo¹, cambiando el nombre de la rama a la hora de subir los archivos:

```
curri@Curro MINGW64 ~/OneDrive/Desktop/EntornoDesarrollo (rama_texto)
$ git push -u origin rama_texto
```

¹ En este ejemplo subiremos la documentación de JavaDoc. El presente escrito, en formato PDF, se subirá en cuanto se terminen los últimos detalles como la bibliografía y el índice.

5.5. Unificación de rama Master con rama `rama_texto` en repositorio local (opcional).

Para unificar todo nuestro contenido a la rama Master, volvemos a esta rama para “actualizar” los cambios que hemos realizado anteriormente (los archivos subidos con `rama_texto`). Esto lo haremos con ayuda del comando “`git merge`” seguido de los nombres `rama_` y Master:

```
curri@Curro MINGW64 ~/OneDrive/Desktop/EntornoDesarrollo (rama_texto)
$ git checkout master
Switched to branch 'master'
Your branch is up to date with 'origin/master'.

curri@Curro MINGW64 ~/OneDrive/Desktop/EntornoDesarrollo (master)
$ git merge rama_texto master
Updating df70fcd..9d1e08b
Fast-forward
 JavaDoc_Apuesta/allclasses-index.html      | 69 +
 JavaDoc_Apuesta/allpackages-index.html     | 65 +
 JavaDoc_Apuesta/element-list                | 2 +
 JavaDoc_Apuesta/help-doc.html              | 187 +++
 JavaDoc_Apuesta/index-files/index-1.html    | 74 +
 JavaDoc_Apuesta/index-files/index-2.html    | 64 +
 JavaDoc_Apuesta/index-files/index-3.html    | 64 +
 JavaDoc_Apuesta/index-files/index-4.html    | 64 +
 JavaDoc_Apuesta/index-files/index-5.html    | 68 +
 JavaDoc_Apuesta/index-files/index-6.html    | 64 +
 JavaDoc_Apuesta/index.html                 | 26 +
 JavaDoc_Apuesta/jquery-ui.overrides.css     | 34 +
 JavaDoc_Apuesta/legal/ADDITIONAL_LICENSE_INFO | 1 +
 JavaDoc_Apuesta/legal/ASSEMBLY_EXCEPTION    | 1 +
 JavaDoc_Apuesta/legal/LICENSE               | 1 +
 JavaDoc_Apuesta/legal/jquery.md             | 72 +
 JavaDoc_Apuesta/legal/jqueryUI.md           | 49 +
```

Una vez enlazados, los añadimos de nuevo a Master y lo confirmamos:

```
curri@Curro MINGW64 ~/OneDrive/Desktop/EntornoDesarrollo (master)
$ git add .

curri@Curro MINGW64 ~/OneDrive/Desktop/EntornoDesarrollo (master)
$ commit -m "cambios finales"
bash: commit: command not found

curri@Curro MINGW64 ~/OneDrive/Desktop/EntornoDesarrollo (master)
$ git commit -m "cambios finales"
[master aeae043] cambios finales
1 file changed, 69 insertions(+)
create mode 100644 module-summary.html

curri@Curro MINGW64 ~/OneDrive/Desktop/EntornoDesarrollo (master)
$ |
```

5.6. Demostración de ramas en GitHub.

Finalizando este escrito, vemos como tenemos en nuestro repositorio remoto las dos ramas creadas con los diferentes archivos subidos. Master para los archivos referentes al código; y la rama_texto con los archivos de texto como la documentación JavaDoc:

The screenshot shows the GitHub interface for the repository 'currofb / OptimizacionDocumentacion', which is public. The navigation bar includes links for Code, Issues, Pull requests, Actions, Projects, Wiki, Security, and Insights. A yellow notification bar at the top states 'rama_texto had recent pushes 14 minutes ago'. Below this, the repository's branch status is shown as 'master' with a dropdown arrow, '2 branches', and '0 tags'. Buttons for 'Go to file' and 'Add' are visible. A 'Switch branches/tags' modal is open, featuring a search bar 'Find or create a branch...', tabs for 'Branches' and 'Tags', and a list of branches: 'master' (checked and marked as 'default') and 'rama_texto'. A 'View all branches' link is at the bottom of the modal. The background shows the start of a commit history with 'Commit 1 códigos' and a file 'df70fcd 1 h'.

currofb / OptimizacionDocumentacion Public

<> Code Issues Pull requests Actions Projects Wiki Security Insights

rama_texto had recent pushes 14 minutes ago

master 2 branches 0 tags Go to file Add

Switch branches/tags

Find or create a branch...

Branches Tags

✓ master default

rama_texto



View all branches



df70fcd 1 h

Commit 1 códigos

Commit 1 códigos



Understand your project by adding a README.




 **rama_texto** 

 2 branches  0 tags

[Go to file](#) [Add file](#) [Code](#)

This branch is **1 commit ahead** of master. [Contribute](#)

 **currofpb** commit 2 textos 9d1e08b 22 minutes ago  2 commits

 Apuesta	Commit 1 códigos	1 hour ago
 Apuesta_Actualizada	Commit 1 códigos	1 hour ago
 JavaDoc_Apuesta	commit 2 textos	22 minutes ago

6. Bibliografía

- <https://www.oracle.com/es/technical-resources/articles/java/javadoc-tool.html>
- https://educacionadistancia.juntadeandalucia.es/centros/sevilla/pluginfile.php/980297/mod_resource/content/1/tutorial_git.pdf
- <https://www.youtube.com/watch?v=mCVQgSyjCkI&list=PLQxX2eiEaqby-qh4raiKfYyb4T7WyHsfW>
- <https://github.com/>
- <https://www.xataka.com/basics/que-github-que-que-le-ofrece-a-desarrolladores>
- <https://git-scm.com/downloads>