

# Ejercicio 4

1)

Renombramos variables para que sean mas descriptivas:

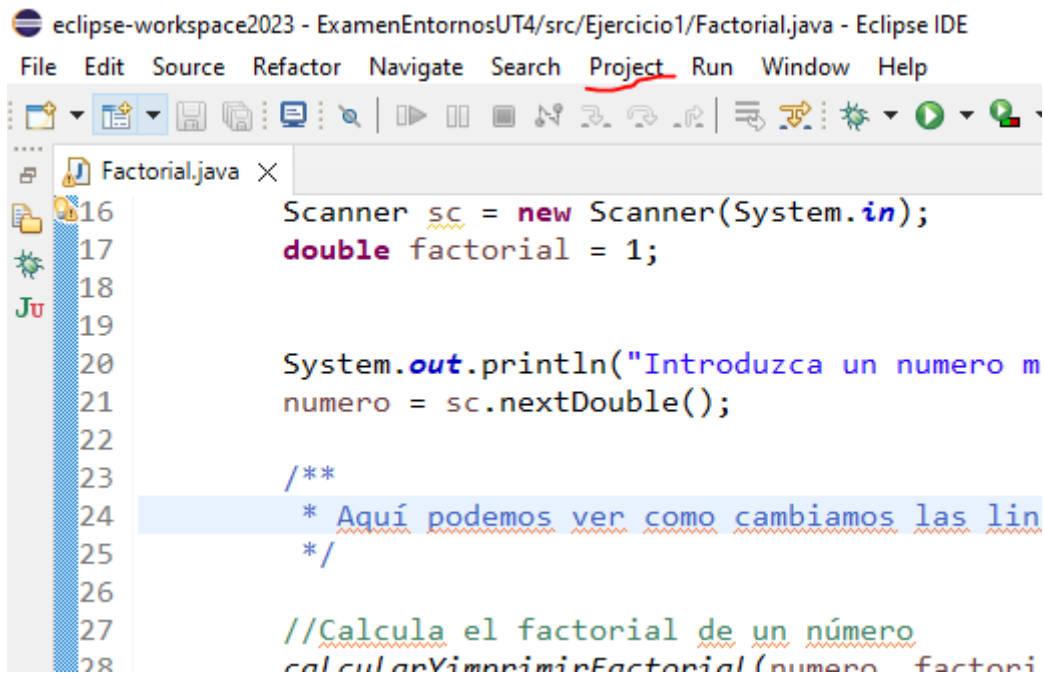
```
public class Factorial {  
  
    public static void main ( String [ ] args ) {  
        double numero;  
        Scanner sc = new Scanner(System.in);  
        double factorial = 1;  
  
        /**  
         *  
         */  
  
        System.out.println("Introduzca un numero mayor o igual a cero: ");  
        numero = sc.nextDouble();  
  
        //Calcula el factorial de un número  
        while ((numero!= 0) && (numero!= 1)) {  
            factorial *= numero;  
            numero--;  
        }  
  
        System.out.println ( factorial ) ;  
    }  
}
```

esta técnica consiste en crear nombres mas específicos para las variables y así poder identificar mejor su uso

La siguiente técnica utilizada seria para las clases muy grandes, en este caso crearemos diversos métodos para hacer el código mas sencillo:

```
double numero;  
Scanner sc = new Scanner(System.in);  
double factorial = 1;  
  
/**  
 *  
 */  
  
System.out.println("Introduzca un numero mayor o igual a cero: ");  
numero = sc.nextDouble();  
  
//Calcula el factorial de un número  
calcularFactorial(numero, factorial);  
  
}  
  
private static void calcularFactorial(double numero, double factorial) {  
    // TODO Auto-generated method stub  
    while ((numero!= 0) && (numero!= 1)) {  
        factorial *= numero;  
        numero--;  
    }  
    imprimirFactorial(factorial);  
}  
  
private static void imprimirFactorial(double factorial) {  
    System.out.println ( factorial ) ;  
}  
  
}
```

Tras haber comentado el código utilizando JAVADOC, crearemos dicha documentación ,para generar la documentación del código deberemos abrir dicho código y dirigirnos a la parte superior de eclipse para poder acceder al apartado de “Project”:



eclipse-workspace2023 - ExamenEntornosUT4/src/Ejercicio1/Factorial.java - Eclipse IDE

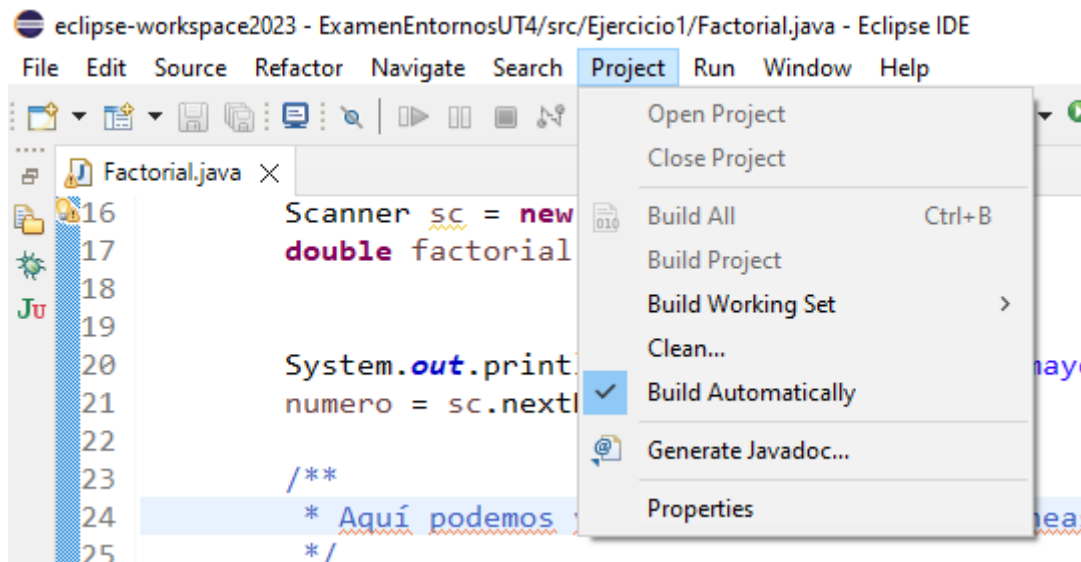
File Edit Source Refactor Navigate Search **Project** Run Window Help

```

16 Scanner sc = new Scanner(System.in);
17 double factorial = 1;
18
19
20 System.out.println("Introduzca un numero m
21 numero = sc.nextDouble();
22
23 /**
24  * Aquí podemos ver como cambiamos las lin
25  */
26
27 //Calcula el factorial de un número
28 calcularYimprimirFactorial(numero, factori


```

Tras esto iremos al apartado de “Generate javadoc” :



eclipse-workspace2023 - ExamenEntornosUT4/src/Ejercicio1/Factorial.java - Eclipse IDE

File Edit Source Refactor Navigate Search **Project** Run Window Help

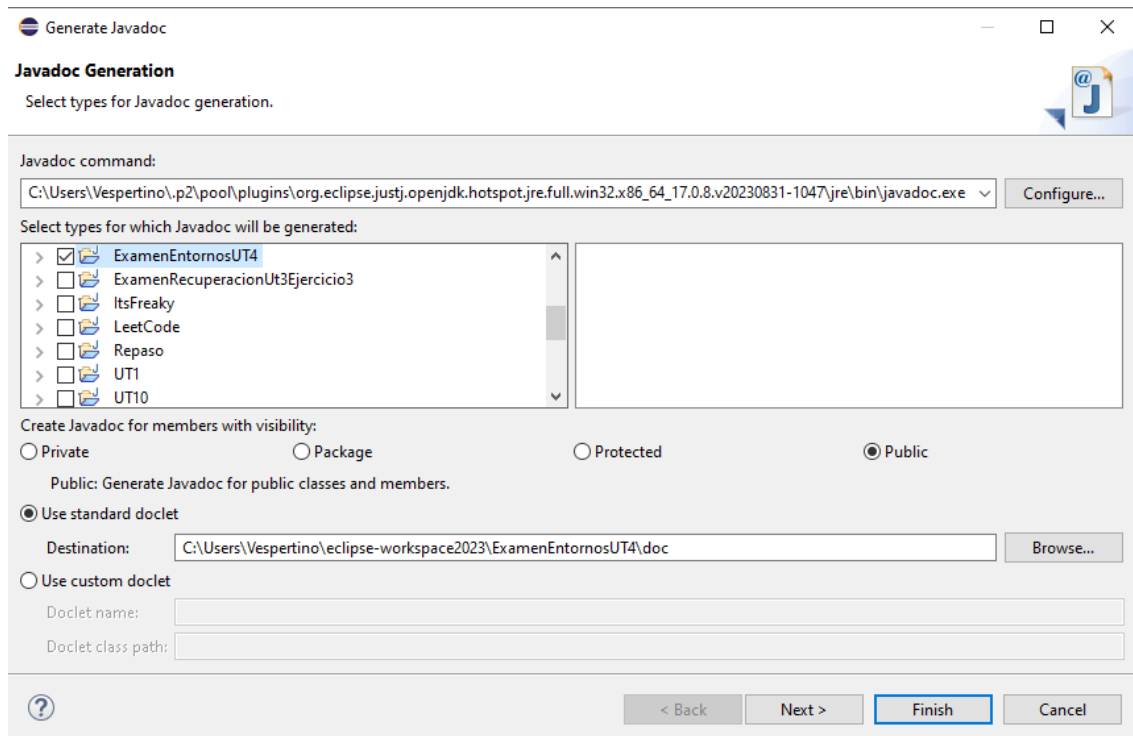
- Open Project
- Close Project
- Build All Ctrl+B
- Build Project
- Build Working Set >
- Clean...
- ☒ Build Automatically
-  Generate Javadoc...
- Properties

```

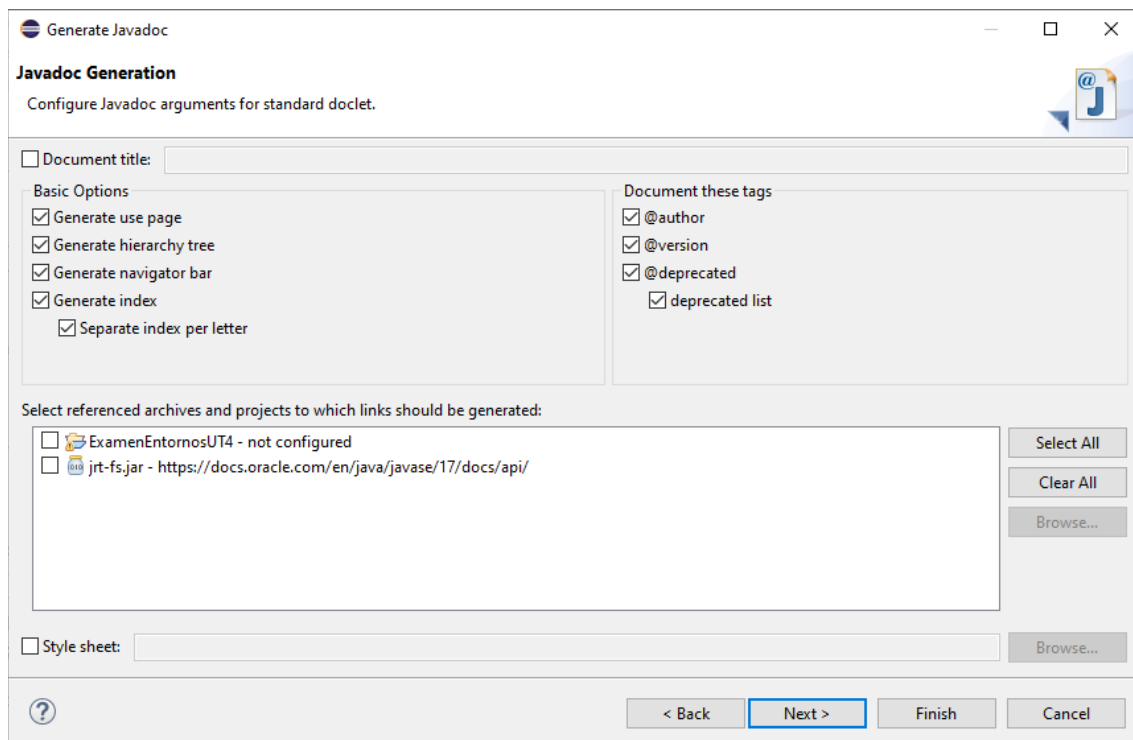
16 Scanner sc = new Scanner(System.in);
17 double factorial = 1;
18
19
20 System.out.println("Introduzca un numero m
21 numero = sc.nextDouble();
22
23 /**
24  * Aquí podemos ver como cambiamos las lin
25  */

```

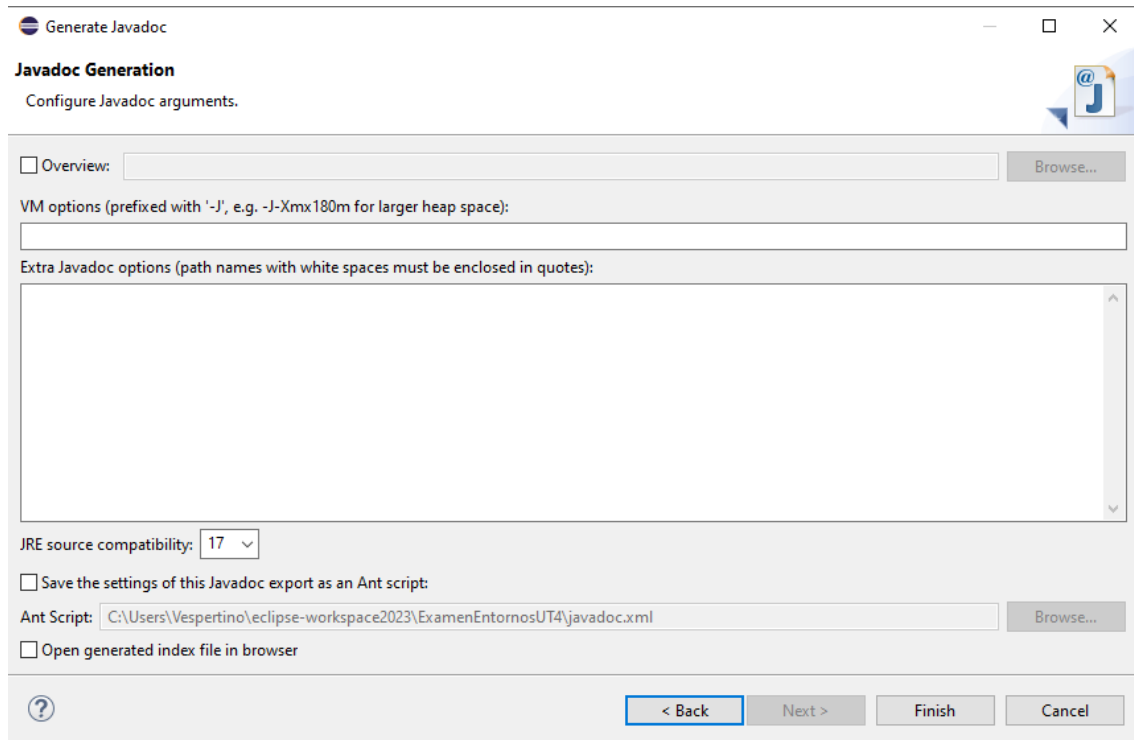
Con esto se nos abrirá una ventana en la cual debemos seleccionar los proyectos que quieres que se genere documentación:



Con esto ya seleccionado le daremos a “next” donde veremos una serie de opciones para la documentación :



Tras esto solo nos quedara una ventana mas en la cual nos dejara añadir alguna configuración extra que nosotros veamos necesaria:



Generate Javadoc

**Javadoc Generation**  
Configure Javadoc arguments.

☐ Overview:

VM options (prefixed with '-J', e.g. -J-Xmx180m for larger heap space):

Extra Javadoc options (path names with white spaces must be enclosed in quotes):

JRE source compatibility:

☐ Save the settings of this Javadoc export as an Ant script:  
Ant Script:

☐ Open generated index file in browser

Con esto ya visto, solo tendremos que darle al botón de “Finish” y para ver la documentacion ya creada.

## Ejercicio 5

a) Responder a la siguiente pregunta que nos hacen en el email: ¿Que es el análisis de código estático? ¿qué es PMD?

El análisis de código estático es aquel tipo de revisiones que se producen en un código que no se va a modificar, el PMD es una herramienta de análisis de código

b) Aplica al menos tres técnicas diferentes de refactorización que hayamos visto en clase al código de "GestorEmpleados.java".

Asegúrate de utilizar la técnica de "Lista de parámetros larga" como una de las técnicas.

Explica brevemente cada técnica utilizada y cómo has aplicado estas técnicas específicamente en partes del código.

Al tener una lista de parámetros muy larga decidimos crear distintos métodos para realizarlo de forma mas sencilla y limpia:

```
public void procesarEmpleado(String nombre, String apellido, int edad, double salario, String departamento, int id, String direccion,
int año=12;
codigoEmpleado(id);

imprimirDetallesEmpleado(nombre, apellido, SalarioAnual(salario, año), bonificacion(salario), departamento);
double salarioTotal = salario * año + bonificacion(salario);
System.out.println("Salario Total Anual: " + salarioTotal);
}

private double bonificacion(double salario) {
// TODO Auto-generated method stub
double bonificacion = salario * 0.1;
return bonificacion;
}

private double SalarioAnual(double salario, int año) {
// TODO Auto-generated method stub
double salarioAnual = salario * año;
return salarioAnual;
}

private int codigoEmpleado(int id) {
// TODO Auto-generated method stub
int codigoEmpleado;
return codigoEmpleado = id;
}
```

Ademas de este reemplazamos números mágicos con una constante simbólica como es en este caso el numero doce que lo cambiamos por una variable llamada año:

```
public void procesarEmpleado(String nombre, String apellido, int edad, double salario, String departamento, int id, String direccion,
int año=12;
codigoEmpleado(id);

imprimirDetallesEmpleado(nombre, apellido, SalarioAnual(salario, año), bonificacion(salario), departamento);
double salarioTotal = salario * año + bonificacion(salario);
System.out.println("Salario Total Anual: " + salarioTotal);
}
```

Y además podemos apreciar que hay una variable que almacena datos que no usamos, en este caso se trata del código de empleado por lo que lo eliminamos:

```
public void procesarEmpleado(String nombre, String apellido, int edad, double salario, String departamento, int id, String direccion,
    int año=12;

    imprimirDetallesEmpleado(nombre, apellido, SalarioAnual(salario, año), bonificacion(salario), departamento);
    double salarioTotal = salario * año + bonificacion(salario);
    System.out.println("Salario Total Anual: " + salarioTotal);
}

private double bonificacion(double salario) {
    // TODO Auto-generated method stub
    double bonificacion = salario * 0.1;
    return bonificacion;
}

private double SalarioAnual(double salario, int año) {
    // TODO Auto-generated method stub
    double salarioAnual = salario * año;
    return salarioAnual;
}
```

c) Comentar el código ya refactorizado usando JAVADOC y generar la documentación del proyecto del ejercicio.

```
public class GestorEmpleados {
    /**
     * @author Álvaro Serradilla Otero
     * Un programa para procesar a los empleados
     * @param nombre
     * @param apellido
     * @param edad
     * @param salario
     * @param departamento
     * @param id
     * @param direccion
     * @param telefono
     * @param email
     */

    public void procesarEmpleado(String nombre, String apellido, int edad, double salario, String departamento, int id, String direccion,
        int año=12;

        imprimirDetallesEmpleado(nombre, apellido, SalarioAnual(salario, año), bonificacion(salario), departamento);
        double salarioTotal = salario * año + bonificacion(salario);
        System.out.println("Salario Total Anual: " + salarioTotal);
    }

    /**
     * Calculamos la bonificacion
     * @param salario
     * @return
     */
    private double bonificacion(double salario) {
        // TODO Auto-generated method stub
        double bonificacion = salario * 0.1;
    }
}
```

## EJERCICIO 6

1) *Aplica al menos tres técnicas diferentes de refactorización (vistas en clase o no). Describe brevemente cada técnica utilizada y justifica cómo contribuyen a mejorar el código.*

La primera que vamos a utilizar es la de clases muy grandes ya que utilizaremos un nuevo método para imprimir los resultados:

```
public class UsoCalculadoraGeometria {  
    public static void main(String[] args) {  
        double r = 5;  
        double l = 10;  
  
        imprimirAreaYvolumen(r,l);  
    }  
  
    private static void imprimirAreaYvolumen(double r, double l) {  
        // TODO Auto-generated method stub  
        System.out.println("Área del círculo: " + aCirculo(r));  
        System.out.println("Volumen del círculo: " + vCirculo(r));  
        System.out.println("Área del cuadrado: " + aCuadrado(l));  
        System.out.println("Volumen del cuadrado: " + vCuadrado(l));  
    }  
}
```

El siguiente que utilizaremos será el de renombrar variables y hacerlas más descriptivas:

```
public class UsoCalculadoraGeometria {  
    public static void main(String[] args) {  
        double radio = 5;  
        double lados = 10;  
  
        imprimirAreaYvolumen(radio,lados);  
    }  
  
    private static void imprimirAreaYvolumen(double radio, double lados) {  
        // TODO Auto-generated method stub  
        System.out.println("Área del círculo: " + aCirculo(radio));  
        System.out.println("Volumen del círculo: " + vCirculo(radio));  
        System.out.println("Área del cuadrado: " + aCuadrado(lados));  
        System.out.println("Volumen del cuadrado: " + vCuadrado(lados));  
    }  
}
```

Por ultimo aremos los métodos mas sencillos y eliminaremos ciertas partes de algunos métodos que se pueden considerar como código muerto:

```
package Ejercicio3;

public class CalculadoraGeometria {

    public static double aCirculo(double r) {
        return Math.PI * r * r;
    }

    public static double vCirculo(double r) {
        return Math.PI * Math.pow(r, 2) * r;
    }

    public static double aCuadrado(double llado) {
        return llado * llado;
    }

    public static double vCuadrado(double lado) {
        double resultado = lado * lado * lado;
        return resultado;
    }
}
```

2) Una vez refactorizado el código, debes documentarlo adecuadamente utilizando JavaDoc. Genera dichos documentos de JavaDoc para el proyecto

```
public class UsoCalculadoraGeometria {
    /**
     *
     * @param args
     */
    public static void main(String[] args) {
        /**
         * @author Alvaro Serradilla Otero
         */
        double radio = 5;
        double lados = 10;

        imprimirAreaYvolumen(radio,lados);
    }
    /**
     *
     * @param radio
     * @param lados
     * Imprime el area y el volumen de un circulo y de un cuadrado
     */
    private static void imprimirAreaYvolumen(double radio, double lados) {
        // TODO Auto-generated method stub
        System.out.println("Area del círculo: " + aCirculo(radio));
        System.out.println("Volumen del círculo: " + vCirculo(radio));
        System.out.println("Area del cuadrado: " + aCuadrado(lados));
        System.out.println("Volumen del cuadrado: " + vCuadrado(lados));
    }
}
```