

Actividad 5.1 – Refactorización, Documentación y GIT



Imagen generada por Gemini IA

Alumno: David García Pasamar
Profesor: Salvador Bordón Ruano
Asignatura: Entornos de Desarrollo (ETS)
Curso: 1º DAM, grupo B (nocturno)

Contenido

Enunciado	3
Ejercicio 1	3
Ejercicio 2	6
Curiosidad	10
Ejercicio 3	10
Apartado a.....	10
Apartado b.....	12
Apartado c.....	13
Apartado d.....	14
Apartado e.....	15
Apartado f.	15
Apartado g.....	17
Apartado h.....	18
Apartado i.....	19
Apartado j.....	19
Apartado k.....	22
Apartado l.....	23
Anexo Apartado l.....	23
Apartado m.	24
Anexo Ejercicio 3	25
Ejercicio 4	25
Ejercicio 5	26
Anexo GitHub	28
Conclusiones	29
Referencias.....	30

Enunciado

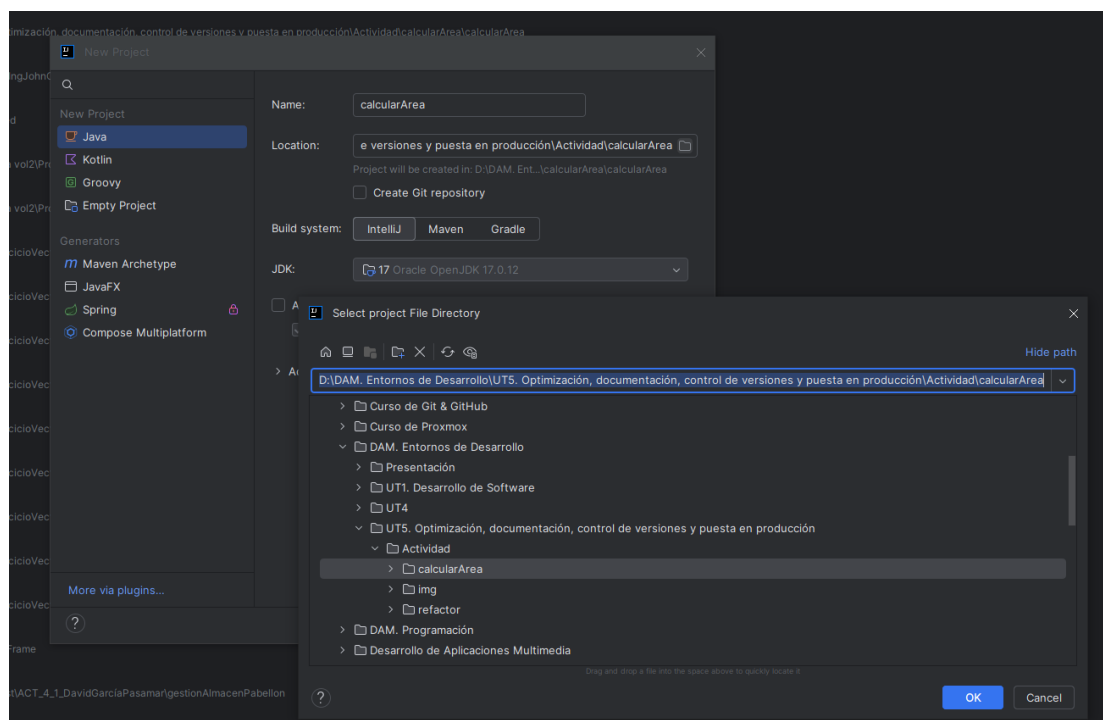
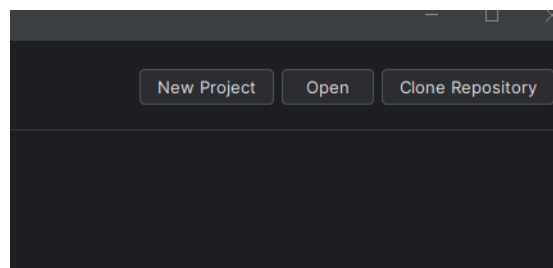
Haciendo uso de los ficheros contenidos en el fichero adjunto (refactor.rar) realizar las siguientes tareas:

Ejercicio 1

Crear un proyecto en IntelliJ IDEA llamado “calcularArea”, añadir un paquete “utilidades” y dentro crear/copiar la clase Circulo, y crear un paquete “figuras” y dentro crear/copiar la clase Test.

Establecer la clase test como la clase principal del proyecto.

1. Comenzamos creando el proyecto

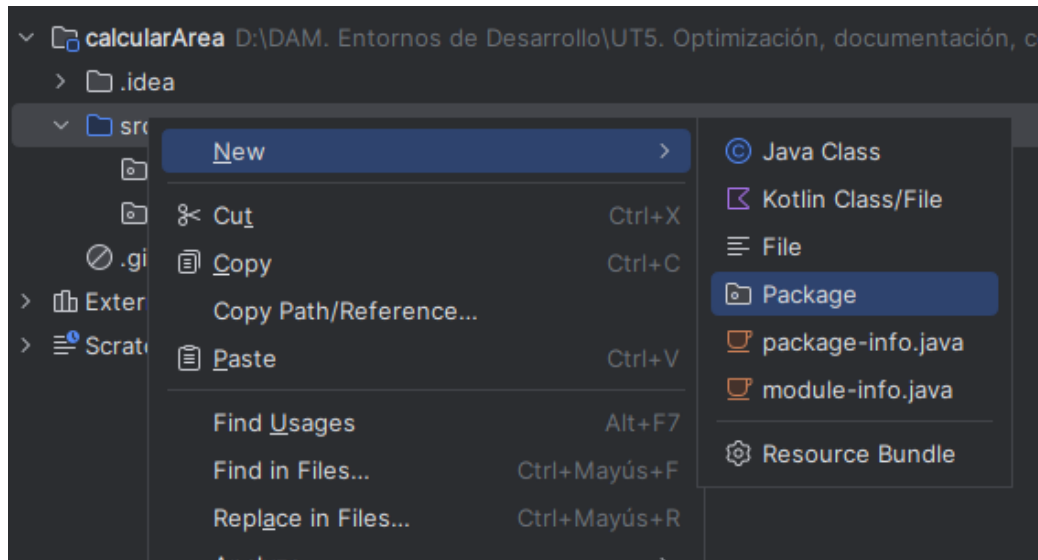


Como vemos en las capturas de pantalla: el proceso es sencillo y lineal, le damos el nombre que se nos solicita, y a continuación le indicamos la carpeta o directorio en el que vamos a crear dicho proyecto.

No marcamos ni Maven ni Gradle, ya que no se nos indica y dejamos indicado al sistema que cree el proyecto por defecto de nuestro IDE (IntelliJ).

Como JDK utilizaremos **OpenJDK 17**, ya que es una versión muy estable y ampliamente utilizada por la industria, además de ser de código abierto.ⁱ

2. Añadir paquete “utilidades”

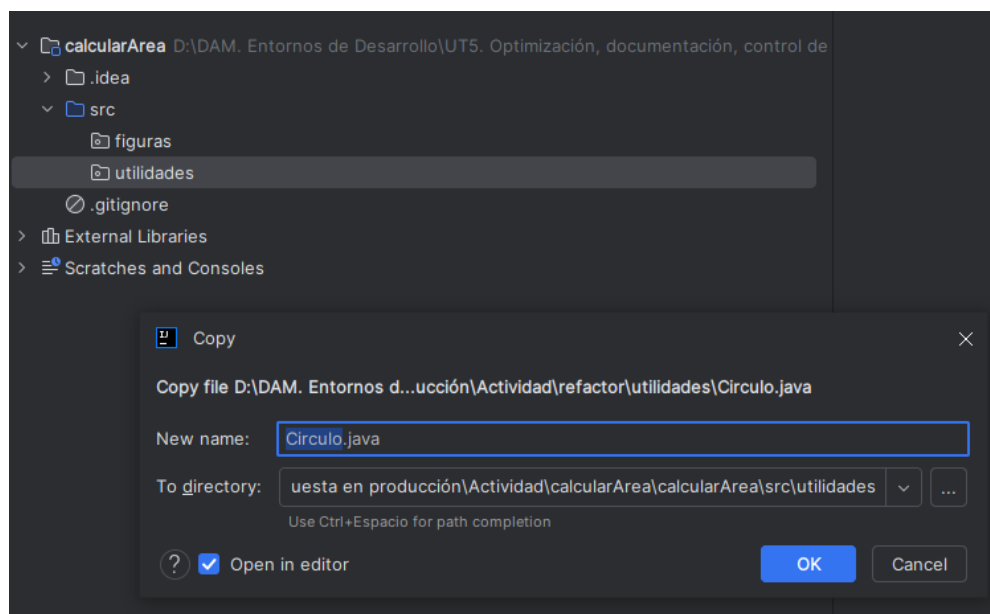


En la carpeta src (source), pinchamos con el botón derecho del ratón, y seguimos la ruta indicada en la imagen. Al clicar en Package nos aparece otra pantalla en la cual le damos nombre al paquete.

3. Crear/Copiar la clase Circulo

Buscamos en el explorador de Windows la carpeta en la que descomprimos el archivo descargado (refactor.rar) y damos a copiar con el botón derecho en el archivo Circulo.java.

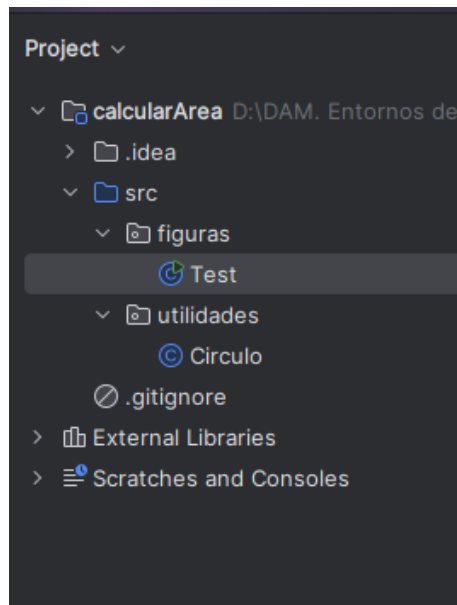
En el IDE, con el botón derecho en el paquete indicado, indicamos paste (pegar) y pegamos el archivo en el lugar correspondiente.



4. Añadir paquete figuras y dentro crear/copiar la clase Test

De manera similar a los dos apartados anteriores, repetimos el proceso.

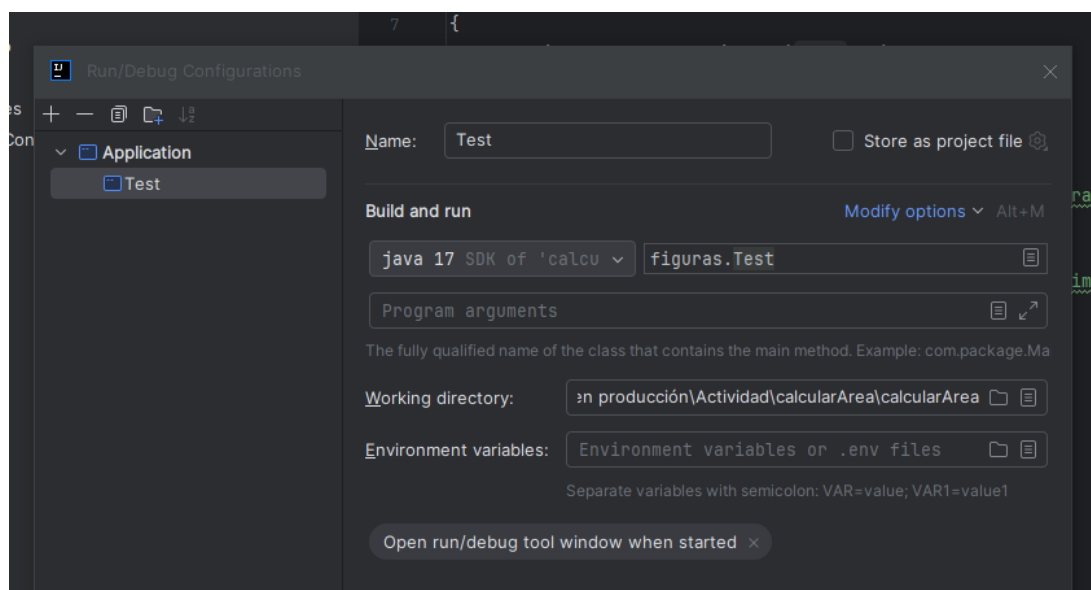
Podemos comprobar la estructura de paquetes y también vemos como el IDE nos está indicando que la clase Test.java contiene el main



5. Configurar la clase Test como la principal del proyecto.

Nos desplazamos a la pestaña **Run** del IDE y buscamos **Edit Configurations...** tal como nos muestra la guía del trabajo.

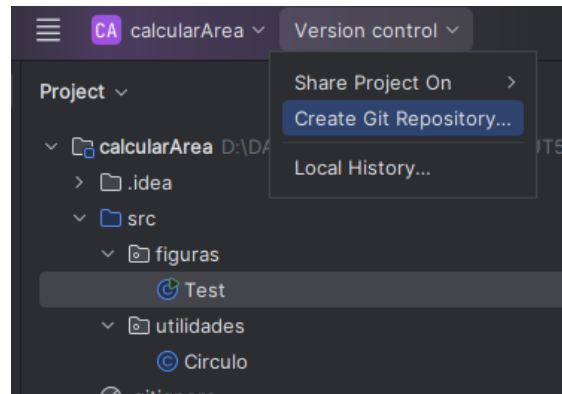
En ella, en la ventana de Configuración pinchamos en el símbolo + (Alt + Insert) e indicamos que queremos configurar una aplicación: le decimos el nombre y el paquete al que pertenece. Tal y como vemos en la imagen siguiente.



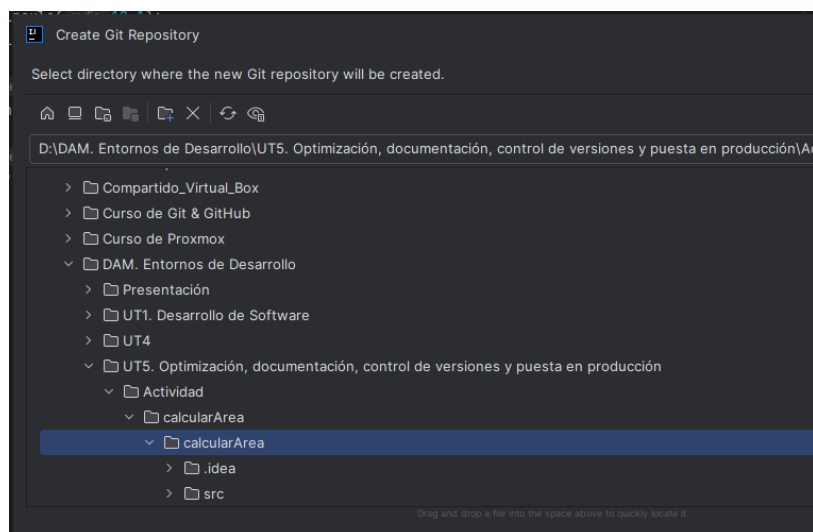
Ejercicio 2

Aplicar Git al proyecto para el control de versiones y realizar el primer commit.

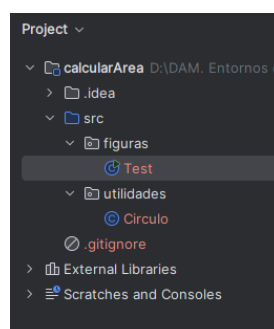
Vamos a la pestaña de Version Control y pinchamos en “Create Git Repository...”



A continuación, le indicamos el directorio desde el cual se creará el repositorio. Nos interesa que pille todo el proyecto, para que cualquier cambio realizado en el mismo nos aparezca marcado en Git.



Al principio nos aparecen los archivos en color rojo, esto es una indicación que Git todavía no se está haciendo cargo de ellos, es decir, no se está realizando el control de cambios todavía.



Esto lo podemos comprobar con los comandos de Git: vamos a abrir la terminal del IDE y con el comando: **git status** veremos que nos muestra Git

```
Terminal Local x
PS D:\DAM. Entornos de Desarrollo\UT5. Optimización, documentación, control de versiones y puesta en producción\Actividad\calcularArea\calcularArea> git status
On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        .gitignore
        .idea/
        src/

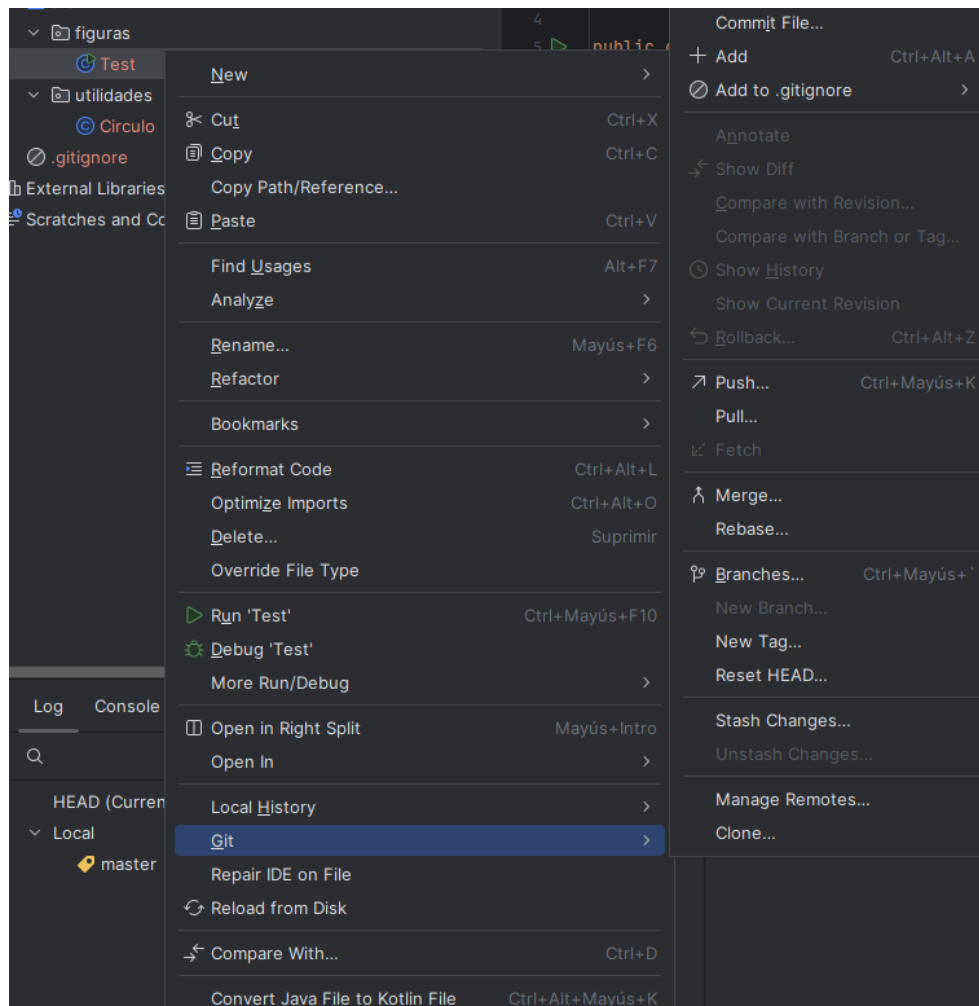
nothing added to commit but untracked files present (use "git add" to track)
PS D:\DAM. Entornos de Desarrollo\UT5. Optimización, documentación, control de versiones y puesta en producción\Actividad\calcularArea\calcularArea>
```

Podemos ver que efectivamente los archivos aparecen como **“Untracked”**.

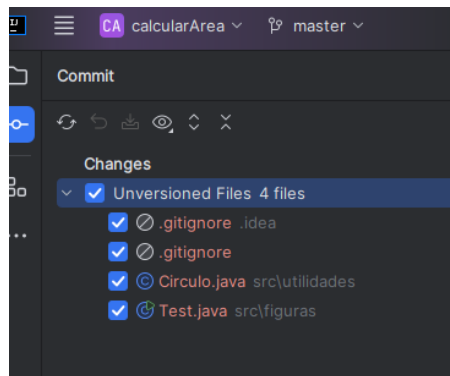
Podemos añadirlos al **“stage”** que es la zona intermedia a falta de confirmación.

Por terminal usamos el comando **git add**.

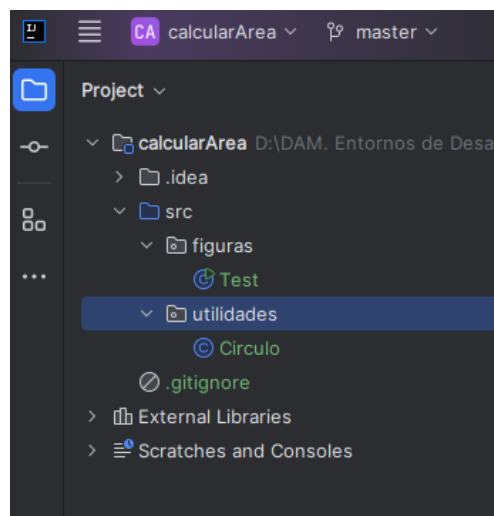
Otra forma desde el IDE es ir al archivo en particular que queremos añadir y marcamos como se muestra en la imagen. **New > Git > Add**



O también yendo a la pestaña commit del lateral y marcando los archivos que queremos añadir al **“stage”**



Con el botón derecho del ratón le indicamos que se añadan al VCS (Version Control System)



Podemos desde la terminal con el comando **git status** ver que se han añadido los distintos archivos al stage y aparecen en color verde.

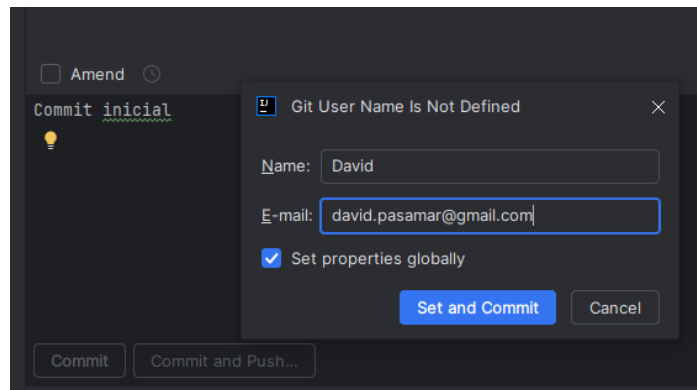
```
Terminal Local x + v
PS D:\DAM. Entornos de Desarrollo\UT5. Optimización, docume
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
    new file:   .gitignore
    new file:   .idea/.gitignore
    new file:   src/figuras/Test.java
    new file:   src/utilidades/Circulo.java

PS D:\DAM. Entornos de Desarrollo\UT5. Optimización, docume
```

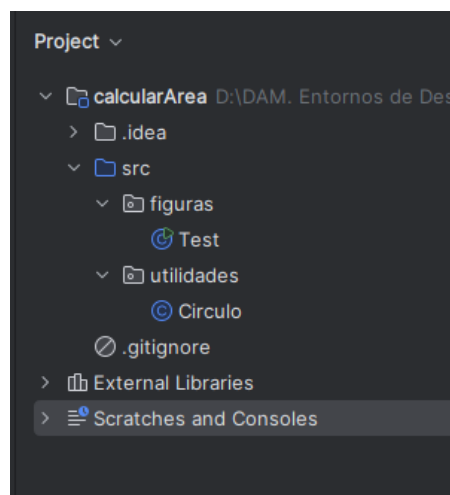
Nos queda realizar un commit para terminar de confirmar los cambios.



En el primer commit tenemos que definir nuestro usuario, con un nombre y un e-mail. Si vemos en la terminal con `git status`, nos dirá que no queda nada por “commitear” y el árbol de trabajo queda limpio.

```
PS D:\DAM. Entornos de Desarrollo\UT5. Optimización, documentación, control de versiones y puesta en producción\Actividad\calcularArea\calcularArea> git status
On branch master
nothing to commit, working tree clean
PS D:\DAM. Entornos de Desarrollo\UT5. Optimización, documentación, control de versiones y puesta en producción\Actividad\calcularArea\calcularArea>
```

Los archivos en el IDE aparecen con su color natural en blanco.



Como curiosidad con el comando ***git log*** podemos ver la evolución de nuestro Git. Vemos en la imagen nuestro commit con un número de identificador para dicho commit, vemos como Git apunta a master y nos comenta el autor y la fecha del commit.

Usando el comando ***git log --oneline --decorate --all --graph*** vemos la cabecera de nuestro commit, a dónde apunta Git y como vemos, nos indica nuestro “***Commit inicial***”.

```
PS D:\DAM. Entornos de Desarrollo\UT5. Optimización, documentación, control de versiones y puesta en producción\Actividad\calcularArea\calcularArea> git log
commit 54b73207aea9159a69dc0b9d0ba3c7b0a174d453 (HEAD -> master)
Author: David <david.pasamar@gmail.com>
Date: Sat Jan 18 21:59:46 2025 +0000

    Commit inicial
PS D:\DAM. Entornos de Desarrollo\UT5. Optimización, documentación, control de versiones y puesta en producción\Actividad\calcularArea\calcularArea> git log --oneline --decorate --all --graph
+ 54b7320 (HEAD -> master) Commit inicial
```



Curiosidad

Realizando el curso de Youtube del canal de MitoCode de Git y GitHub, muestro una captura de terminal, sobre cómo se ve mediante comandos en Linux Mint (versión Cinnamon), una sucesión de commits, etiquetas, y bifurcaciones.ⁱⁱ

```
pasamar@LinuxMint01:~/git/folder 1$ git log --oneline --decorate --all --graph
* 88345a3 (HEAD -> main, tag: v1.0.0, origin/main) Conflictos resueltos
|
| * 9ae0853 (nueva_rama) Se agrego footer
| * 640e99c Se agrego title
|/
* d281b46 Se agrego body
* d6b639b Nueva opcion
* 53d2677 Primer commit de esta rama
* 647a98e Carpeta config
* 187fa6b (tag: v0.0.1) Se agrego gitignore
* 313d86c (tag: alpha_0.1) Se agrego body
* e9705c5 Se cambio de nombre a principal.html
* 528ffde Se agrego principal.html
* 536c26a Nueva modificacion al index
* 8aa26ff Se agrego contenido a la carpeta css
* ee7a5ba Se agrego validacion .js
* 7848b7d Se modifico index.html la palabra Coder
* 33a0e74 Commit Inicial
pasamar@LinuxMint01:~/git/folder 1$
```

Ejercicio 3

Realizar los ajustes de refactorización indicados a continuación. De hecho, si intentas ejecutar la aplicación observarás que se genera un error.

La aplicación debe funcionar correctamente, de no funcionar no se valorará este apartado.

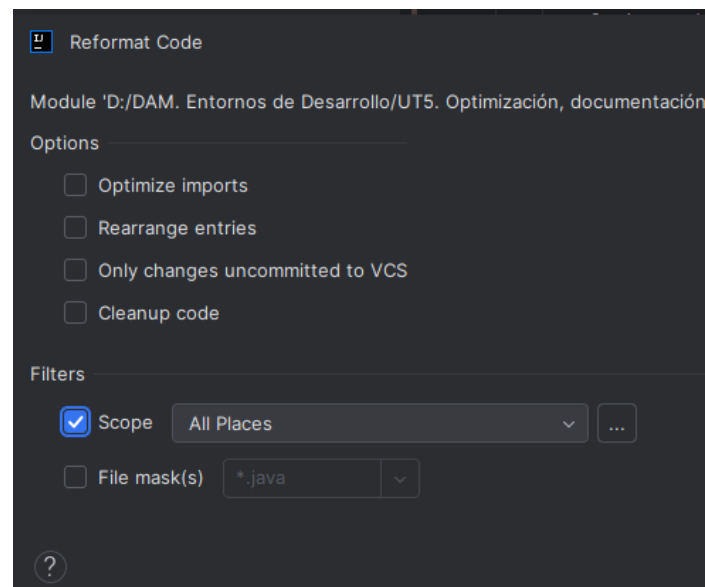
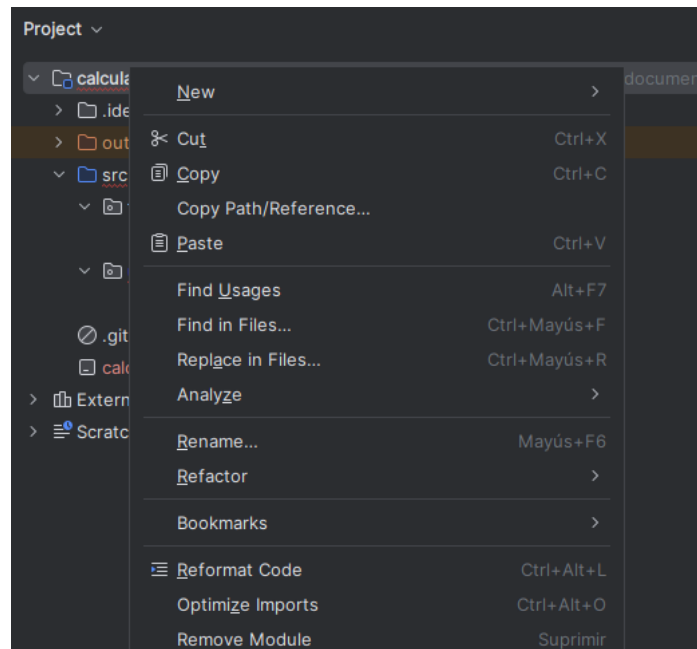
Comprobamos que efectivamente se produce un error al intentar ejecutar la aplicación.

```
D:\DAM. Entornos de Desarrollo\UTS. Optimización, documentación, control de versiones y puesta en producción\Actividad\calcularArea\calcularArea\src\utilidades\Circulo.java:23:33
java: cannot find symbol
  symbol:   method getRad()
  location: variable otro of type utilidades.Circulo
```

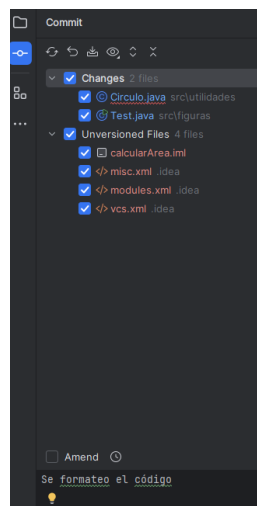
Apartado a.

Corregir la tabulación del código y dar formato al código (Code / Reformar Code)

Vamos al inicio del proyecto, y con el botón derecho pinchamos en **Reformat Code**, y a continuación le indicaremos el scope, que en el primer caso le diremos todo el proyecto.



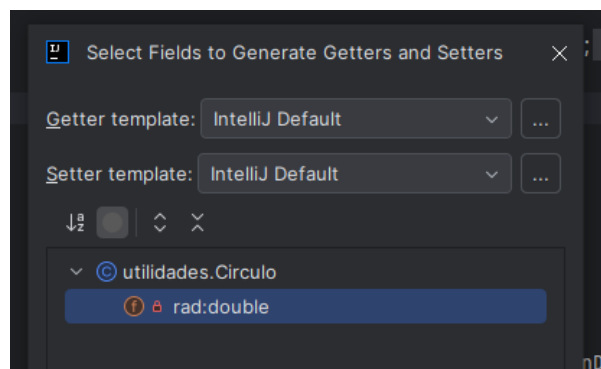
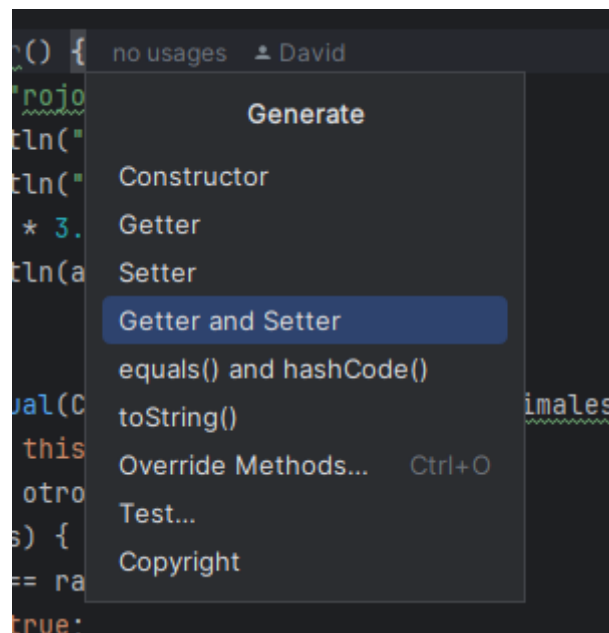
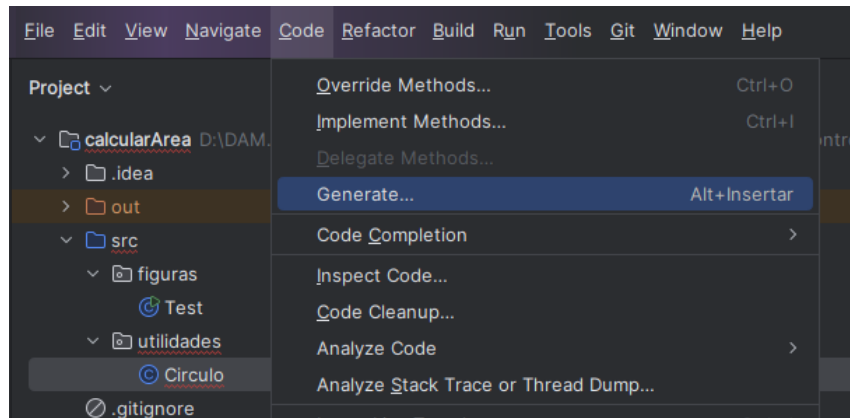
A continuación, actualizamos el repositorio e indicamos con un nuevo commit.



Apartado b.

Generar métodos get y set para la clase Circulo en el atributo rad. Después de crear estos métodos la aplicación funciona correctamente.

Nos dirigimos a la pestaña **Code > Generate > Getter and Setter > rad**
Y vemos como se nos generan ambas funciones.



```
public double getRad() { 1 usage new *
    return rad;
}

public void setRad(double rad) { no usages new *
    this.rad = rad;
}
```

Comprobamos que efectivamente funciona el código y procedemos a actualizar Git e incorporar un nuevo commit.

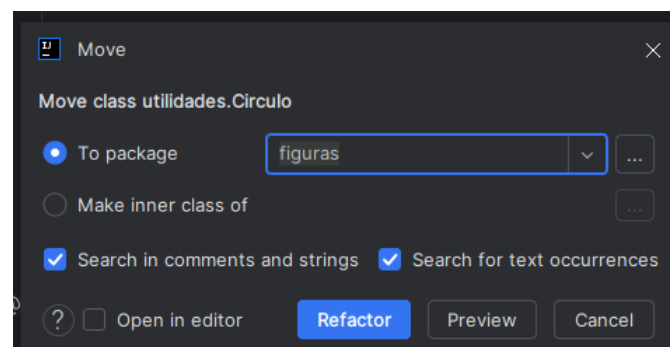
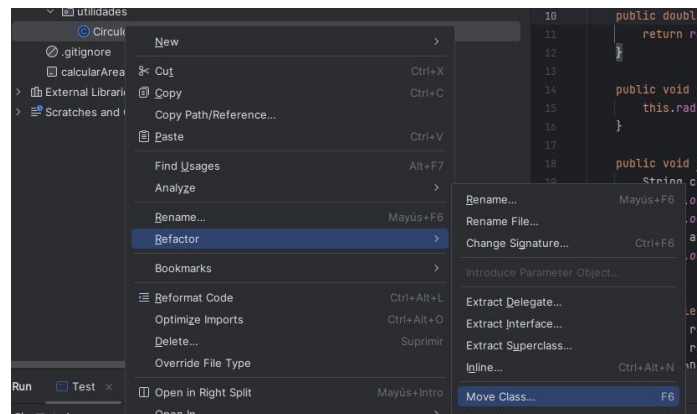
```
Test
" C:\Program Files\Java\jdk-17\bin\java.exe" "-ja
c2 y c3: iguales sin considerar decimales

Process finished with exit code 0
```

Apartado c.

Mover la clase Circulo al paquete figuras.

Realizamos los cambios indicados en las imágenes siguientes:



Igual que en los otros apartados, actualizamos el repositorio.

Apartado d.

Renombrar la clase Circulo por Circunferencia. Observar si el cambio afecta a otras clases (en este caso Test).

Como realizamos en el apartado anterior, nos dirigimos a Refactor pero en esta ocasión elegimos Rename como se ve en las imágenes. Antes de realizar el cambio, pinchamos la opción Preview y vemos cómo afectarán los cambios.

Efectivamente afecta a Test, pero realizamos los cambios en la opción de Refactor y los cambios quedan realizados.

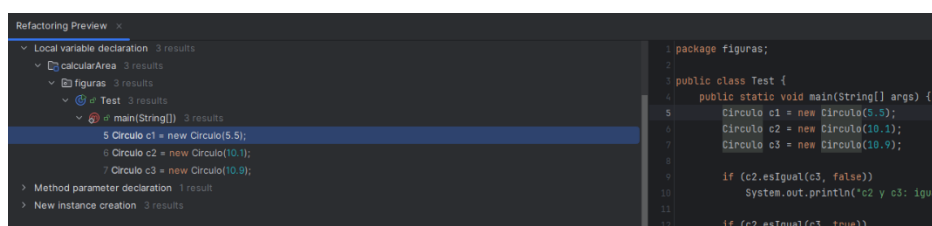
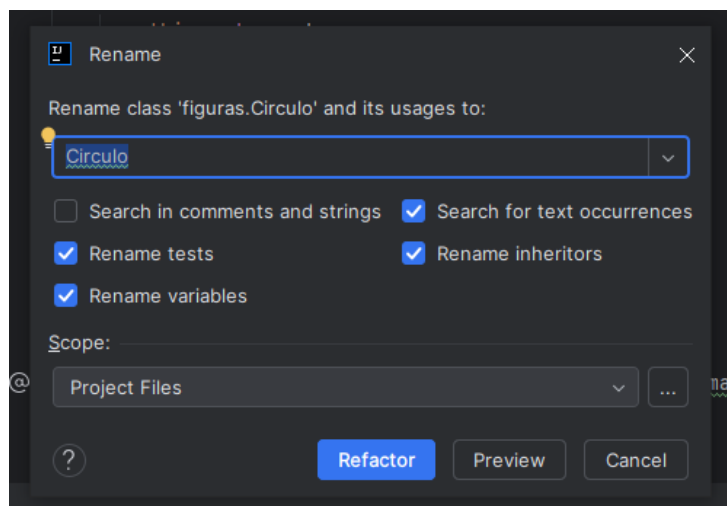
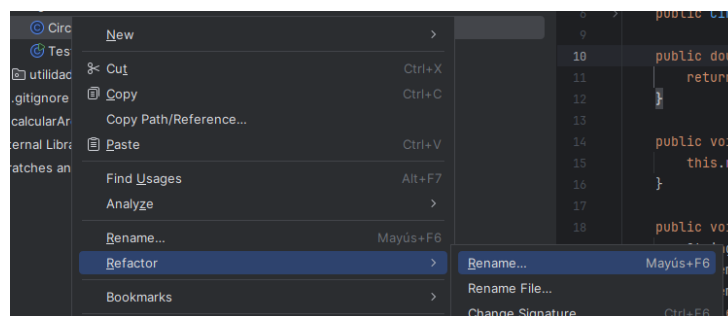
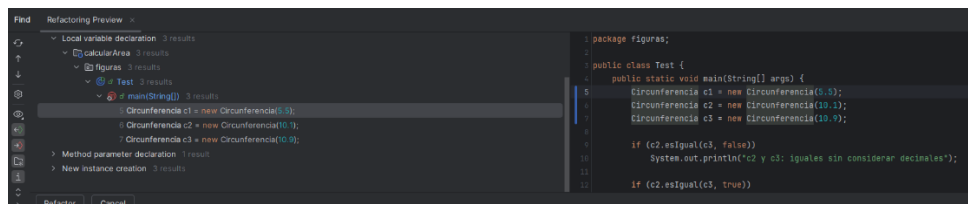


Imagen con los cambios realizados, y como en cada apartado, actualizamos el Git.

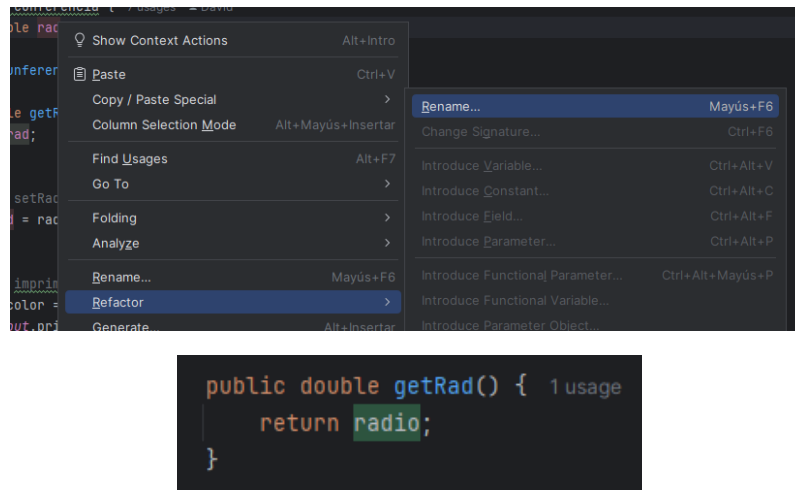


Apartado e.

Renombrar el atributo “rad” por “radio”. ¿Cómo afecta al método get?

De manera similar al cambio de nombre de la clase, realizamos una refactorización al atributo “rad”.

Vemos que el método get actualiza el cambio.



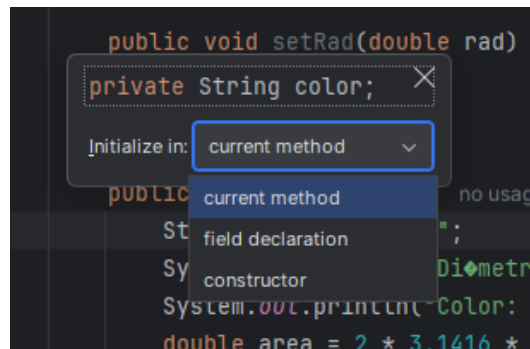
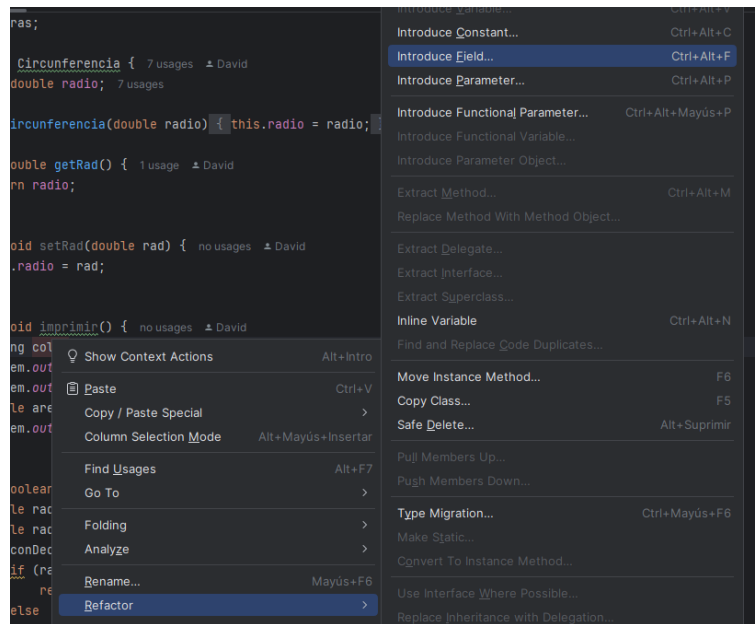
Actualizamos Git.

Apartado f.

Convertir la variable local “color” del método imprimir en un atributo de la clase e inicializando su valor en el mismo método imprimir.

Realizando un paso similar a los anteriores, es decir, pinchando en Refactor, en esta ocasión clickeamos en “Introduce Field...”

En la variable “color” le indicamos que se inicialice en el método en el que estaba y en la última imagen vemos como han quedado los cambios.



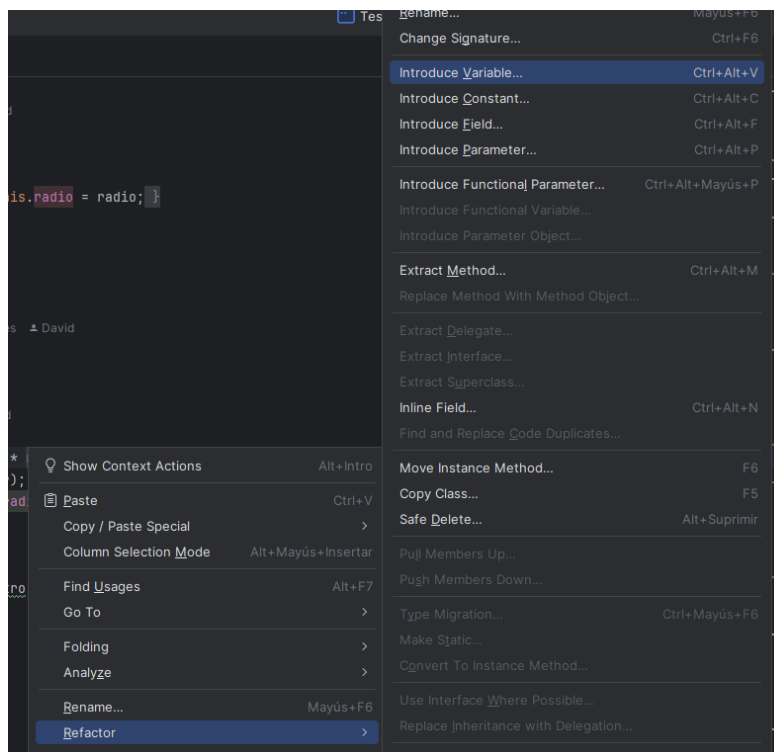
Actualizamos Git.

Apartado g.

En imprimir, en lugar de calcular y escribir el diámetro directamente en el println, extraer a una variable local “d” e imprimir dicha variable.

Seguimos la secuencia como mostramos en las imágenes:

Le damos el nombre “d” a la variable local y procedemos a actualizar Git después de los cambios.



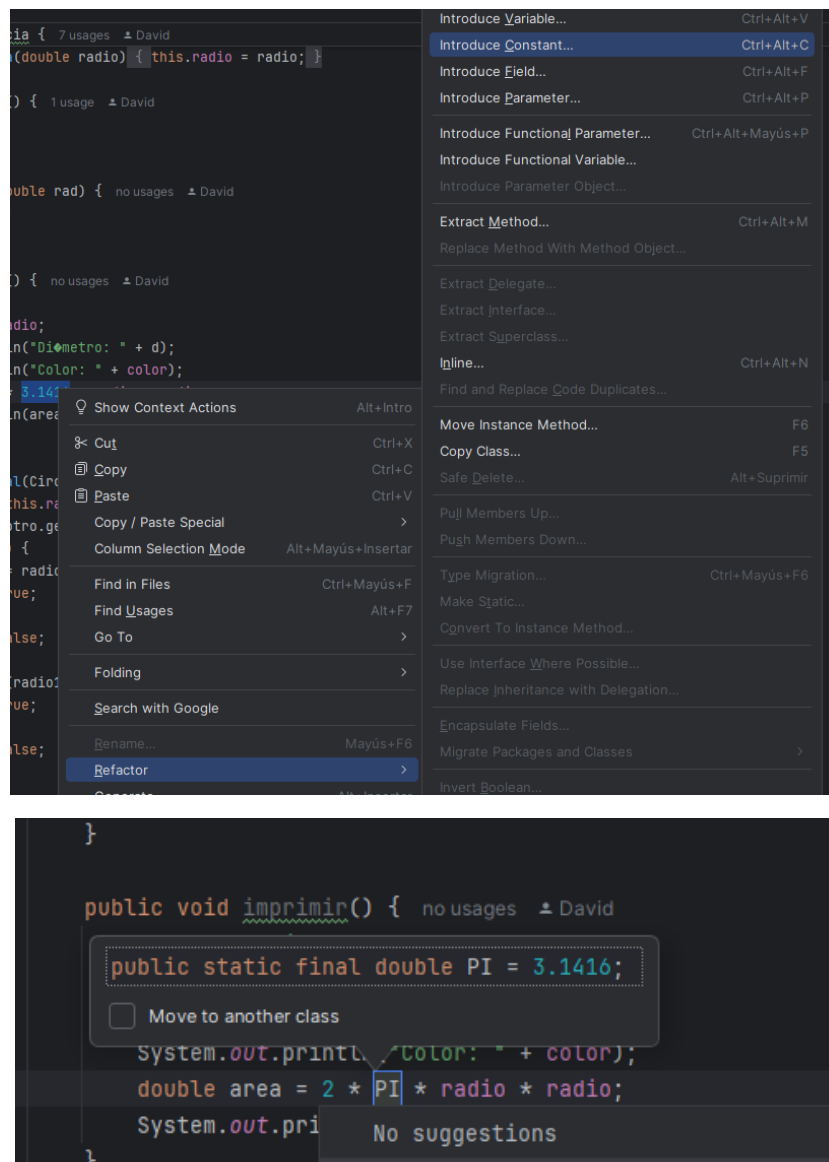
```
public void imprimir() { no usages  David
    color = "rojo";
    double d = 2 * radio;
    System.out.println("Diámetro: " + d);
    System.out.println("Color: " + color);
    double area = 2 * 3.1416 * radio * radio;
    System.out.println(area);
}
```



Apartado h.

Hacer que 3.1416 sea una constante llamada PI

Mostramos los pasos a seguir en las imágenes y actualizamos el Git.





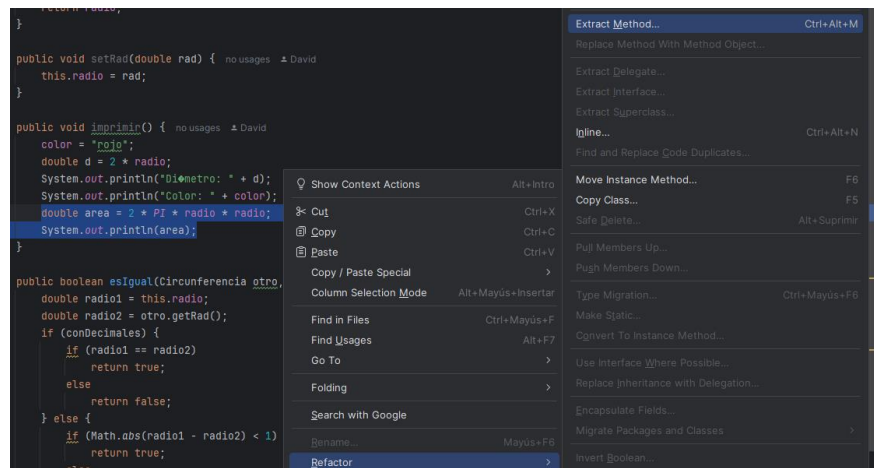
```
public class Circunferencia { 7 usages ± David *
    public static final double PI = 3.1416; 1 usa
    private double radio; 7 usages
    private String color; 2 usages

    public Circunferencia(double radio) { this.ra
```

Apartado i.

Extraer el cálculo del área a un método llamado **calcularArea**.
No recibirá parámetros y devolverá un **double**.

Refactorizamos y Extraemos método:



Indicamos el nombre y comprobamos que la función cumple lo que se ha solicitado.
Finalmente actualizamos Git.

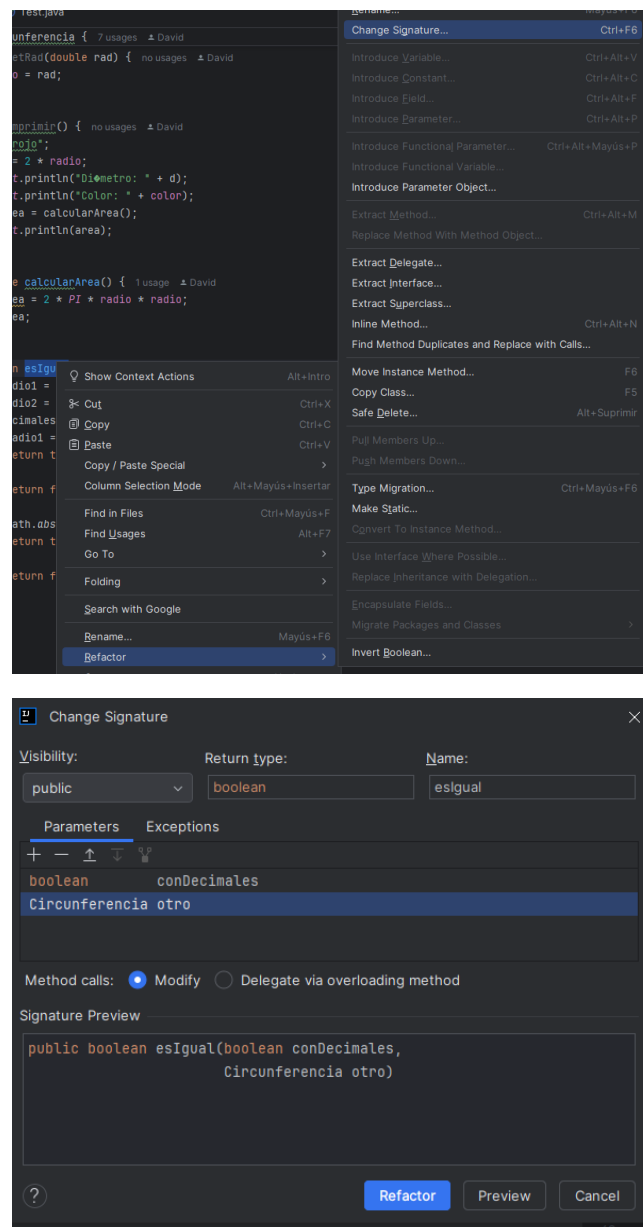
```
System.out.println("Color: " + color);
double area = calcularArea();
System.out.println(area);

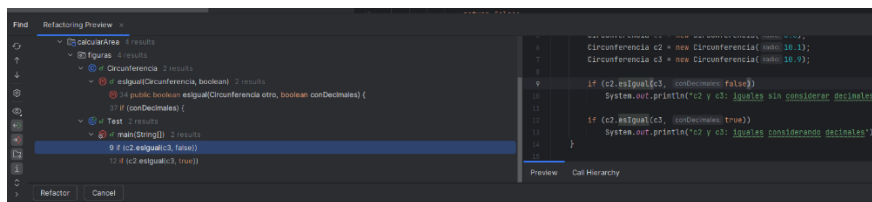
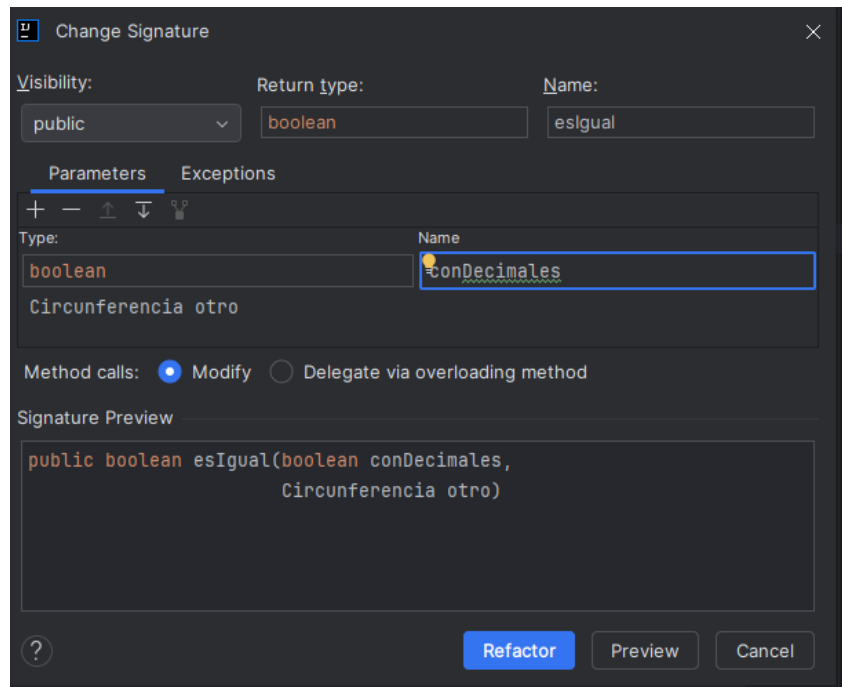
private double calcularArea() {
    double area = 2 * PI * radio * radio;
    return area;
}
```

Apartado j.

Cambiar la firma o cabecera del método esIguar, invirtiendo el orden de los parámetros y cambiando el nombre de conDecimales por considerarDecimales. ¿Cómo afecta el cambio a la clase Test, en la que se usaba este método?

Continuamos explorando dentro de Refactor pero en esta ocasión con **Change Signature**, en el cuadro que nos aparece podemos controlar el orden de los parámetros y si clickeamos en uno de ellos podemos cambiarle el nombre también. Con el botón **Preview** podemos ver a qué clases afectarían los cambios. En la siguiente secuencia de capturas podemos comprobar al final el resultado de los cambios.





```
public boolean esIgual(boolean considerarDecimales, Circunferencia otro) { 2 usage
    double radio1 = this.radio;
    double radio2 = otro.getRad();
    if (considerarDecimales) {
        if (radio1 == radio2)
            return true;
        else
            return false;
    } else {
        if (Math.abs(radio1 - radio2) < 1)
            return true;
        else
            return false;
    }
}
```

```
if (c2.esIgual(considerarDecimales: false, c3))
    System.out.println("c2 y c3: iguales sin considerar decimales");

if (c2.esIgual(considerarDecimales: true, c3))
    System.out.println("c2 y c3: iguales considerando decimales");
```

Como en cada apartado, actualizamos Git.

Al realizar **Run** sobre **Test**, vemos que el resultado sigue siendo el mismo que antes del cambio.

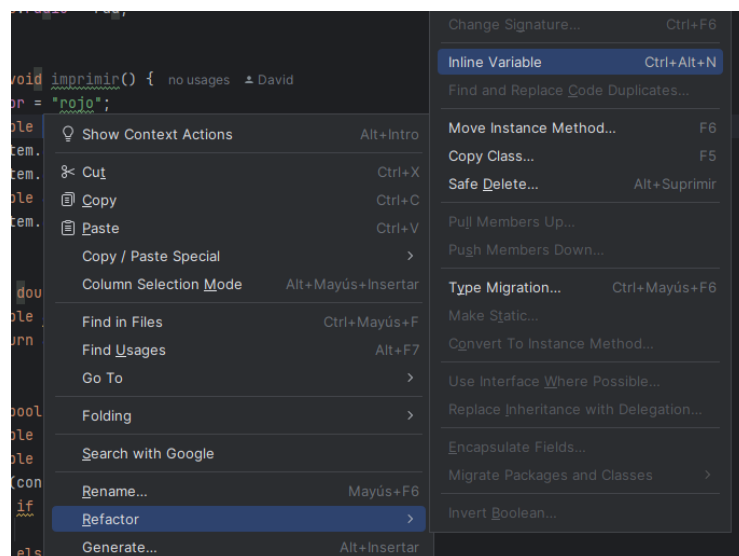
```
Test x
"C:\Program Files\Java\jdk-17\bin\java.exe" "-jav
c2 y c3: iguales sin considerar decimales

Process finished with exit code 0
```

Apartado k.

Ahora se propone usar “inline” para deshacer algunos cambios, es decir, hacer el código más concreto. Seleccionar la variable “d” (diámetro) y hacer que su valor se use en línea, desapareciendo por tanto la variable.

Vamos a **Refactor** y elegimos ahora la opción de **Inline...** como se muestra en las imágenes, veremos a continuación el momento previo al cambio y el posterior de cómo queda ya con la variable desaparecida.



Instante inicial con la variable d, antes del cambio:

```
double d = 2 * radio;
System.out.println("Diámetro: " + d);
System.out.println("Color: " + color);
```

Momento final con la variable desaparecida:

```
color = rojo;
System.out.println("Diámetro: " + 2 * radio);
System.out.println("Color: " + color);
```

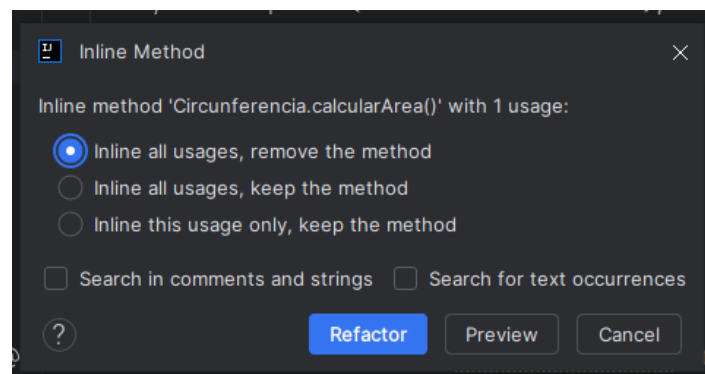
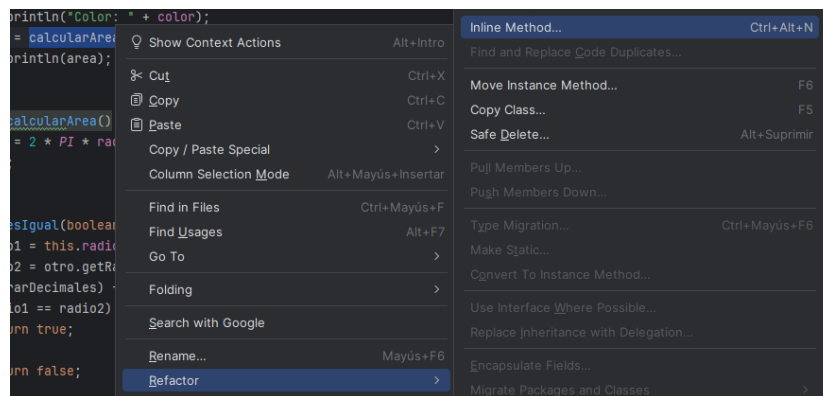
Actualizamos Git.

Apartado I.

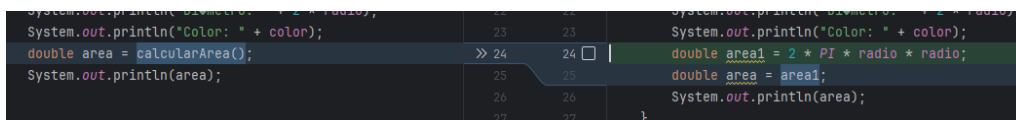
Seleccionar la llamada al método calcularArea y hacer que su código se incorpore en la misma línea, desapareciendo la necesidad de usar el método (se puede borrar el método después)

De manera similar al anterior apartado, ahora realizaremos una refactorización con Inline method.

Veremos que en el cuadro que nos aparece tenemos la opción de eliminar el método a posteriori, así que enseñaremos primeramente como nos cambia la línea de llamada al método y luego manualmente eliminamos el método ya que no se va a usar.

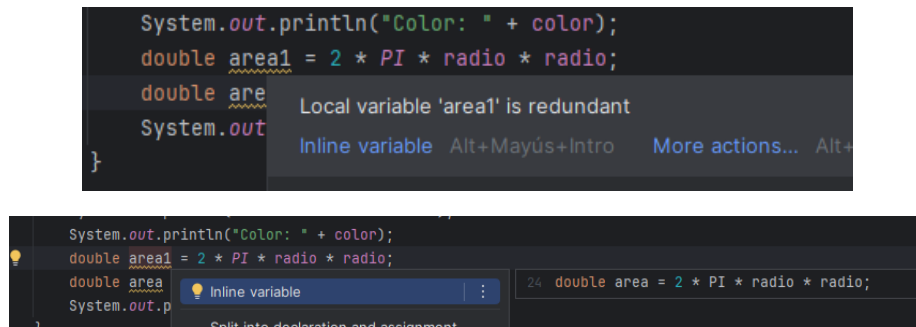


Vemos como hemos modificado la llamada al método, haciendo que el código del método ya no tenga ninguna utilidad.



Anexo Apartado I.

Podemos comprobar a la vista, e incluso nos lo indica el propio IDE, que la variable resultante por defecto de area1 es redundante con area. Con lo cual le indicaremos al IDE que tal y como vimos en el apartado anterior, realice una modificación inline de la variable.



```
System.out.println("Color: " + color);
double area1 = 2 * PI * radio * radio;
double area
System.out
}

System.out.println("Color: " + color);
double area1 = 2 * PI * radio * radio;
double area
System.out.p

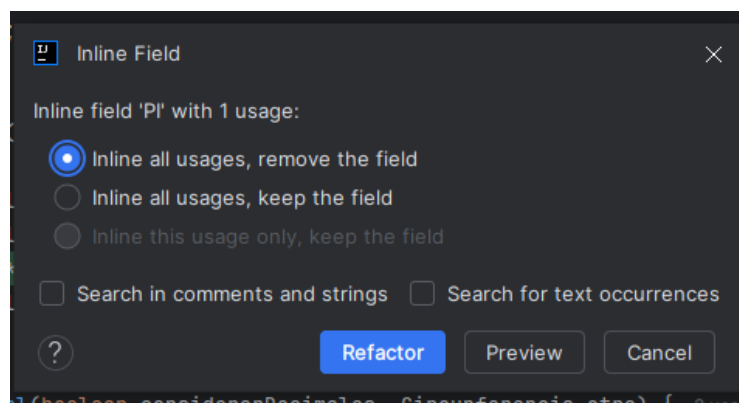
24 double area = 2 * PI * radio * radio;
```

Ahora sí, con todos los cambios realizados, actualizamos Git.

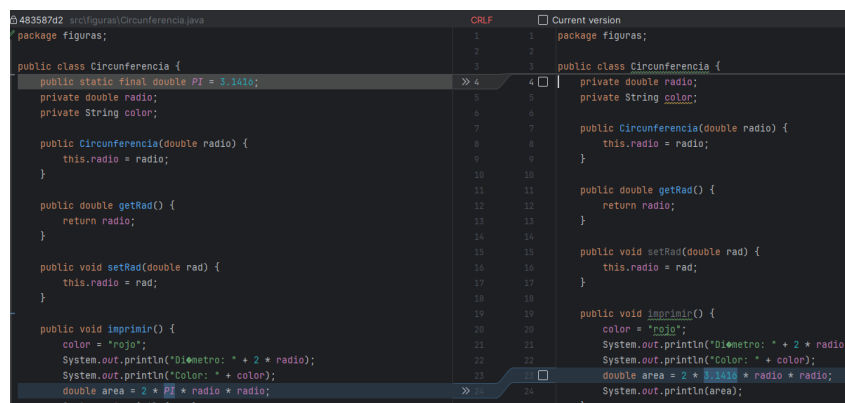
Apartado m.

Seleccionar la constante PI y hacer que su valor se incorpore a las líneas en que se usa, desapareciendo por tanto la constante.

Se igual forma que realizamos con la variable “d” en el apartado k. ahora resolvemos la constante PI



Vemos como quedan los cambios:



```
package figuras;

public class Circunferencia {
    public static final double PI = 3.1416;
    private double radio;
    private String color;

    public Circunferencia(double radio) {
        this.radio = radio;
    }

    public double getRad() {
        return radio;
    }

    public void setRad(double rad) {
        this.radio = rad;
    }

    public void imprimir() {
        color = "rojo";
        System.out.println("Diámetro: " + 2 * radio);
        System.out.println("Color: " + color);
        double area = 2 * PI * radio * radio;
        System.out.println(area);
    }
}

package figuras;

public class Circunferencia {
    private double radio;
    private String color;

    public Circunferencia(double radio) {
        this.radio = radio;
    }

    public double getRad() {
        return radio;
    }

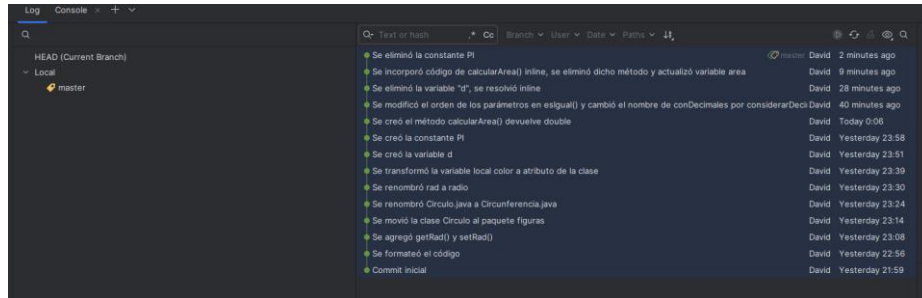
    public void setRad(double rad) {
        this.radio = rad;
    }

    public void imprimir() {
        color = "rojo";
        System.out.println("Diámetro: " + 2 * radio);
        System.out.println("Color: " + color);
        double area = 2 * 3.1416 * radio * radio;
        System.out.println(area);
    }
}
```

Actualizamos Git.

Anexo Ejercicio 3

Veamos como han quedado todos los cambios del ejercicio 3 en nuestro repositorio de Git.



Y de igual forma lo vemos desde el terminal:

```
PS D:\DAM. Entornos de Desarrollo\UTS. Optimización, documentación, control de versiones y puesta en producción\Actividad\calcularArea\calcularArea> git log
* ba10dce (HEAD -> master) Se eliminó la constante PI
* fca8bd5 Se incorporó código de calcularArea() inline, se eliminó dicho método y actualizó variable area
* d0a39cb Se eliminó la variable "d", se resolvió inline
* be32f44 Se modificó el orden de los parámetros en esIgual() y cambió el nombre de conDecimales por considerarDecimales.
* c8c8f24 Se creó el método calcularArea() devuelve double
* 2dc159c Se creó la constante PI
* f3c76fb Se creó la variable d
* a03a0ba Se transformó la variable local color a atributo de la clase
* b0b1a9c Se renombró rad a radio
* 0a45ad2 Se renombró Circulo.java a Circunferencia.java
* dc5791a Se movió la clase Circulo al paquete figuras
* 751a1a3 Se agregó getRad() y setRad()
* c01769d Se formateó el código
* 54b7320 Commit inicial
```

Ejercicio 4

Documentar el contenido del proyecto y adjuntar la documentación generada por javaDoc.

Siguiendo los apuntes y leyendo alguna web de ayudaⁱⁱⁱ, además de revisar por encima la documentación oficial de Oracle^{iv}. Lo primero que realizamos son los comentarios en las clases y métodos.

Ejemplo de cómo estamos incorporando los comentarios:

```
/**
 * La clase Circunferencia representa una circunferencia con los atributos
 * Contiene métodos para imprimir en consola su color, diámetro y calcular
 */
public class Circunferencia { 7 usages  ▲ David
    private double radio; 7 usages
    private String color; 2 usages

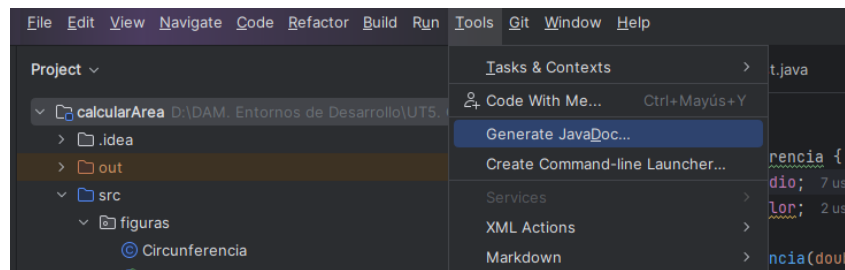
    /**
     * Método que crea una circunferencia con un radio dado
     *
     * @param radio El parámetro radio define el radio de la circunferencia
     */
    public Circunferencia(double radio) { 3 usages  ▲ David
        this.radio = radio;
    }

    /**
     * Método que obtiene el radio de la circunferencia
     *
     * @return devuelve el radio de la circunferencia
     */
    public double getRad() { 1 usage  ▲ David
        return radio;
    }

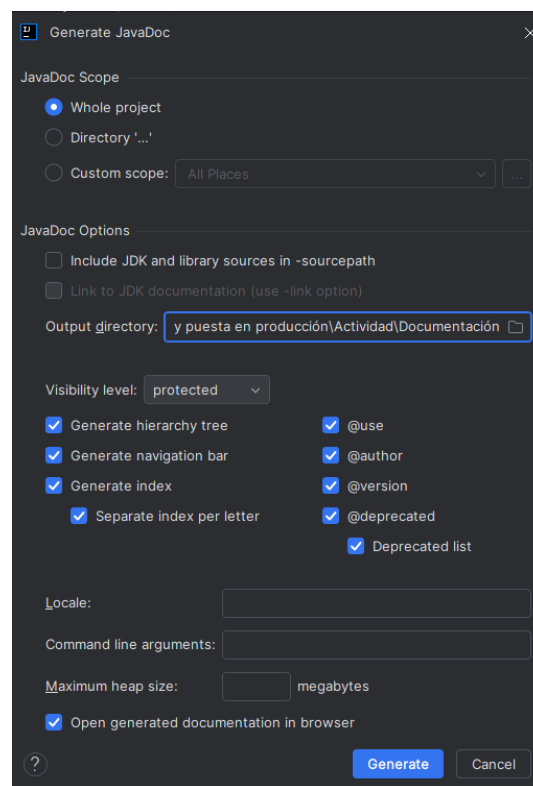
    /**
     * Método que establece el radio de la circunferencia
     */
}
```



Una vez realizados los comentarios, vamos a la pestaña **Tools > Generate JavaDoc**



Marcamos todas las opciones, y dado que estamos realizando un manual para clase, entendemos que sería un manual para el equipo de desarrollo, con lo cual, el nivel de visibilidad deseado lo dejamos en **protected**, que nos incluye a todos los parámetros del proyecto.



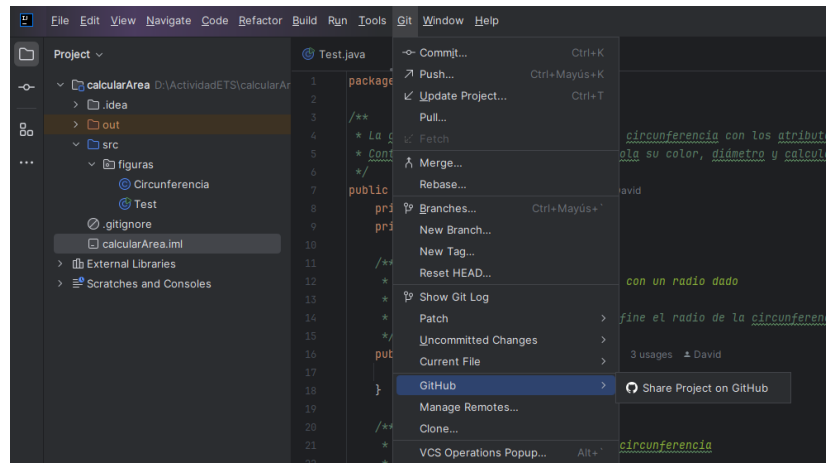
Y generamos nuestro documento de ayuda.

Como hemos añadido comentarios al proyecto, actualizamos el repositorio con un nuevo commit.

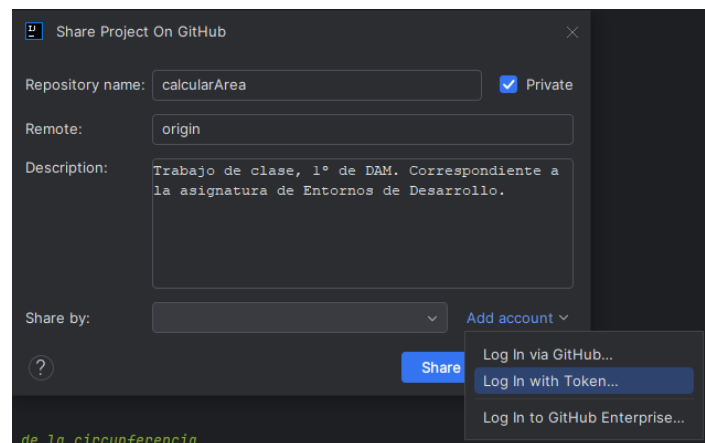
Ejercicio 5

Sincronizar el proyecto con una cuenta de GitHub. Indicar el enlace para poder descargarlo y probarlo.

En el propio IDE, dentro de la pestaña Git, vemos que hay un apartado para GitHub.

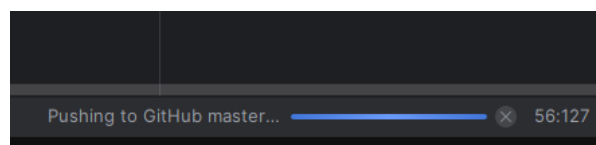


Al pinchar nos aparece una nueva ventana para la configuración del repositorio en GitHub. En nuestro caso como ya tenemos un Token para subir el proyecto, elegimos esa opción.

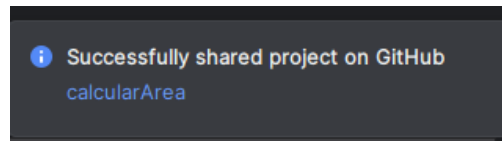


Nos aparece ya el campo Share by: completo y le damos a Share.

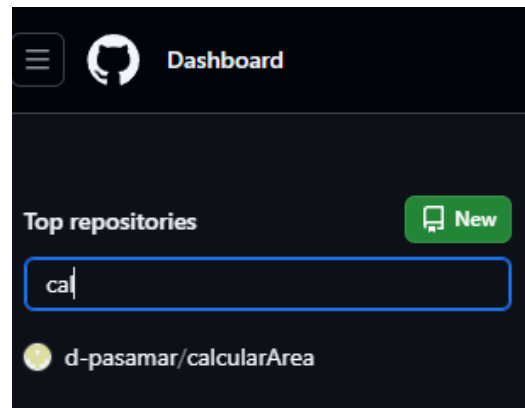
Esperamos un momentito.



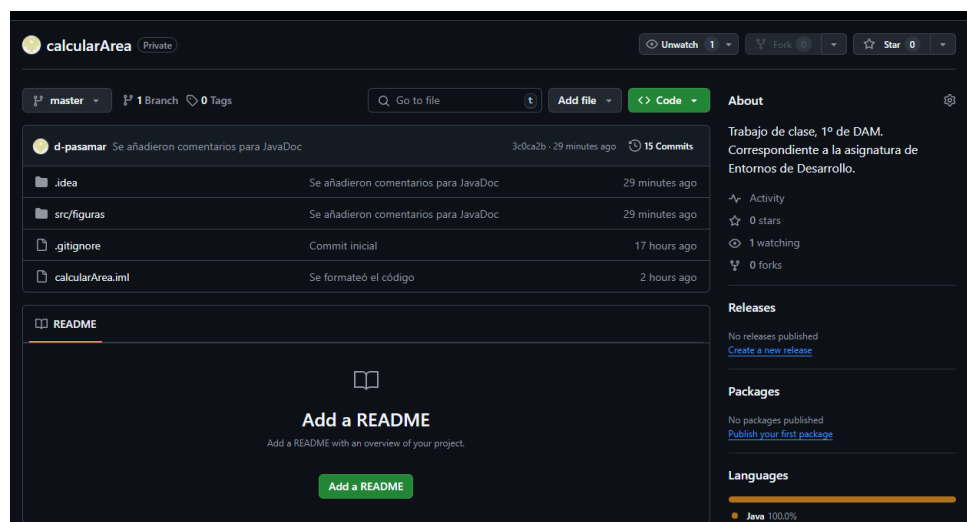
Y vemos el mensaje de éxito



Comprobamos en nuestra cuenta de GitHub:



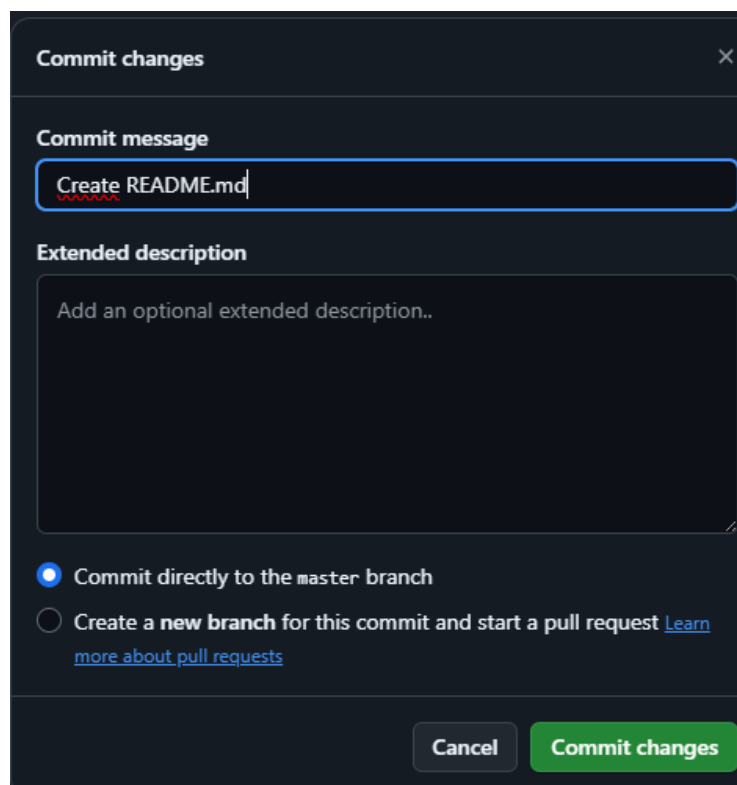
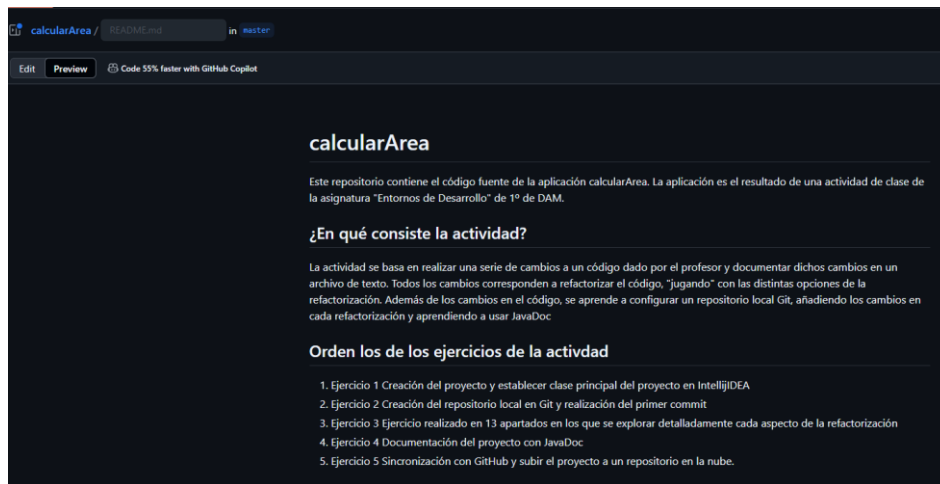
Y vemos el resultado:



Vamos a añadir un readme y presentar el repositorio para compartir.

Anexo GitHub

Para entender mejor la índole del proyecto, es recomendable la realización de una descripción en el repositorio. Siguiendo las indicaciones de la sintaxis de escritura de la web oficial^v vamos completando nuestro readme.



Para completar el trabajo, crearé una carpeta docs en el repositorio donde subiremos el manual creado con JavaDocs y el trabajo en .pdf

El enlace del repositorio en GitHub es:

<https://github.com/d-pasamar/calcularArea>

Conclusiones

Muchas veces comenzamos tutoriales de cómo programar en un lenguaje u otro. Recorremos los mismos temas: tipos de datos, secuencias de control, bucles, clases, interfaces gráficas, pero en cuanto notamos que hacemos nuestros pequeños programas, nos damos por satisfechos.



Es ahí cuando uno investiga un poco más profundamente, cuando vemos que nos queda mucho por aprender y es en este punto dónde creo que esta actividad tiene bastante importancia. Ya que enfoca distintos aspectos que deberíamos conocer: Encapsulamiento, saber refactorizar, reformartear un código, aprender el uso de Git, y derivando en GitHub. Documentación con JavaDoc.

Sin duda este trabajo nos sirve como referencia rápida de cómo refactorizar, reformatear y trabajar con Git y GitHub con IntelliJIDEA

Referencias

ⁱ TodoCode. “Curso gratis Java para principiantes 2025”. Youtube.

<https://www.youtube.com/watch?v=qxXcl56NfnE&list=PLQxX2eiEaqbz8W1qM9eAxPSF65Zj090-P> (14:15 – 15:50)

ⁱⁱ MitoCode, “Curso de Git y GitHub”. Youtube:

<https://www.youtube.com/watch?v=j8CSUPIB8mA>

ⁱⁱⁱ Aprenderaprograma.com, “Documentar proyectos Java con Javadoc.”

https://www.aprenderaprogramar.com/index.php?option=com_content&view=article&id=646:documentar-proyectos-java-con-javadoc-comentarios-simbolos-tags-deprecated-param-etc-cu00680b&catid=68&Itemid=188

^{iv} Oracle, “How to Write Doc Comments for the Javadoc Tool”.

<https://www.oracle.com/es/technical-resources/articles/java/javadoc-tool.html>

^v GitHub, “Sintaxis de escritura y formatos básicos”. <https://docs.github.com/es/get-started/writing-on-github/getting-started-with-writing-and-formatting-on-github/basic-writing-and-formatting-syntax>