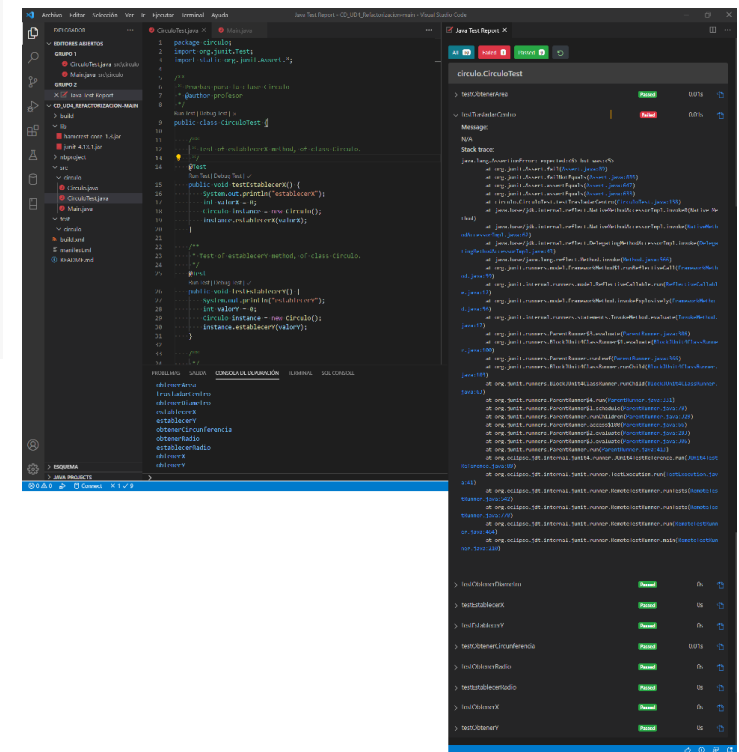


REFACTORIZACIÓN Y DOCUMENTACIÓN

1. REFACTORIZACIÓN

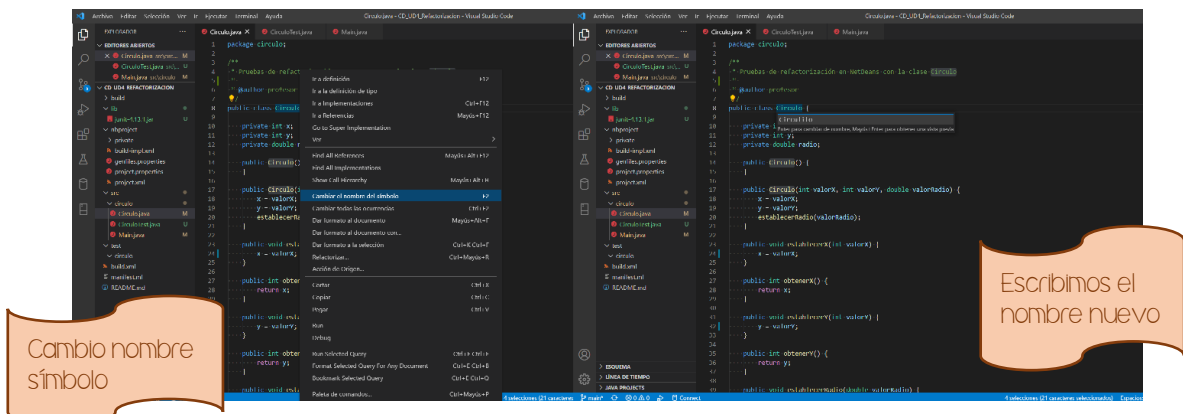
Antes de empezar con la refactorización, probamos la clase test para llevar a cabo las pruebas unitarias del proyecto Círculo y comprobar si el código escrito es correcto.



Se observa un error en el Test "Trasladar centro" que nos indica que el resultado esperado para ese test era 6, sin embargo, el resultado obtenido es 5

A. RENOMBRAR CLASE CÍRCULO POR CIRCULITO:

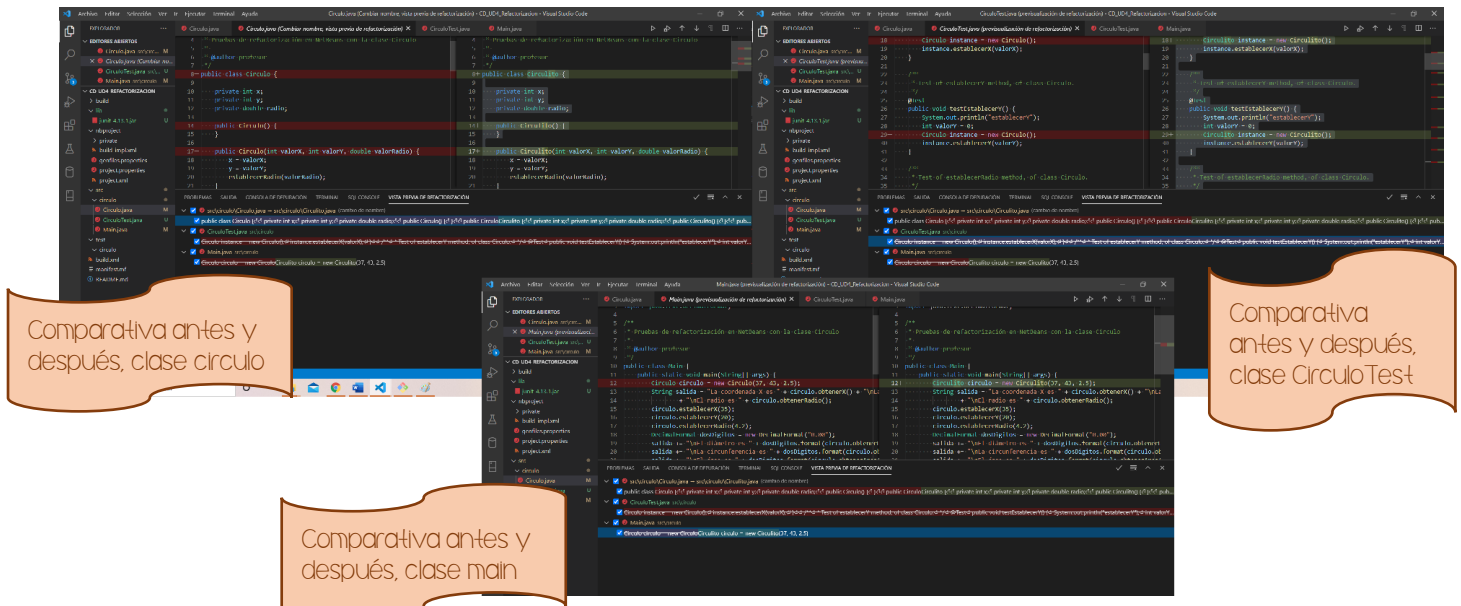
Para cambiar el nombre a la clase, situamos el cursor encima del nombre de la clase y clicamos con el botón derecho. Seleccionamos la opción “Cambiar el nombre del símbolo”, y nos saldrá un recuadro donde escribiremos el nuevo nombre.



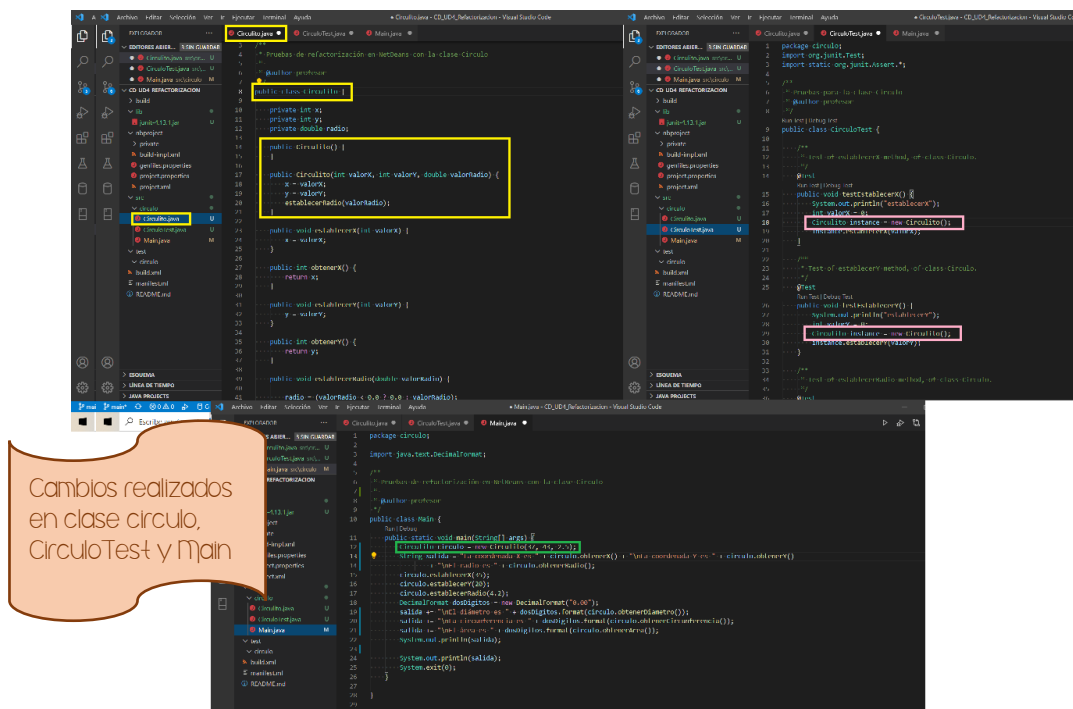
ED 4 – Tarea 4 Entornos de Desarrollo

Alumna: López Diéguez, Silvia

Para ver dónde se aplicarán los cambios, marcamos las teclas shift y enter en el teclado y nos saldrá la siguiente imagen, en la que se aprecia que los cambios que se van a aplicar se harán tanto en la clase “Circulo” como en las clases “Main” y “CirculoTest”. En la parte derecha de cada pantalla se observa el resultado de los cambios y en la parte izquierda el estado original.

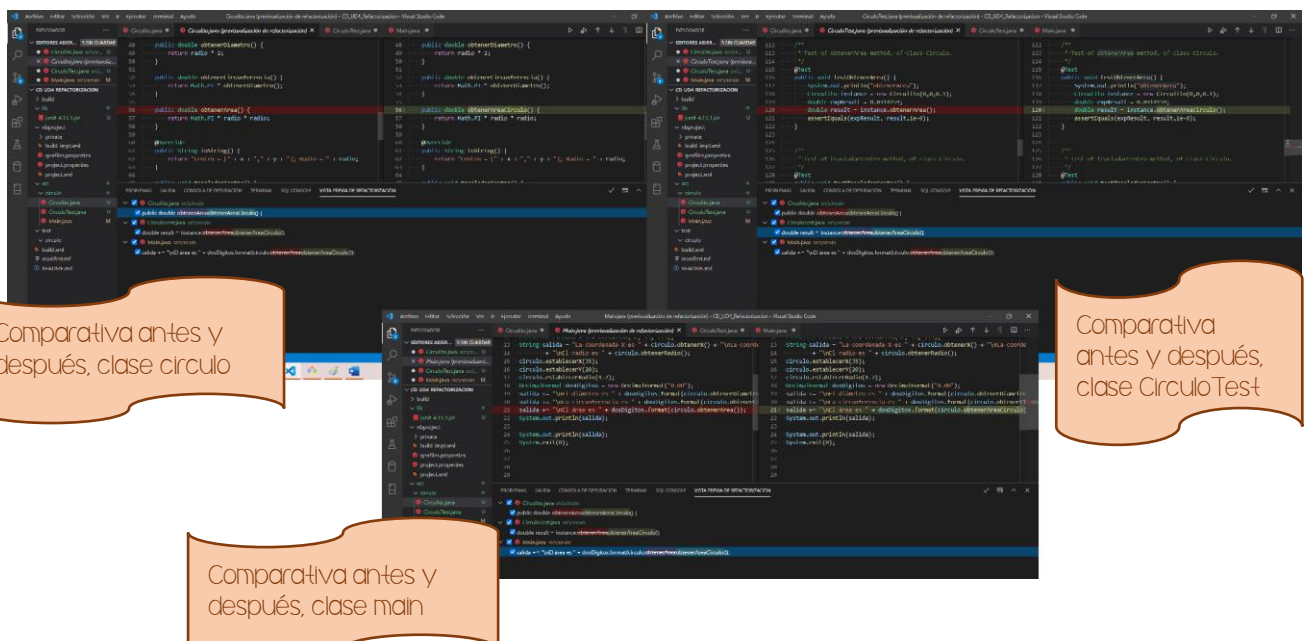
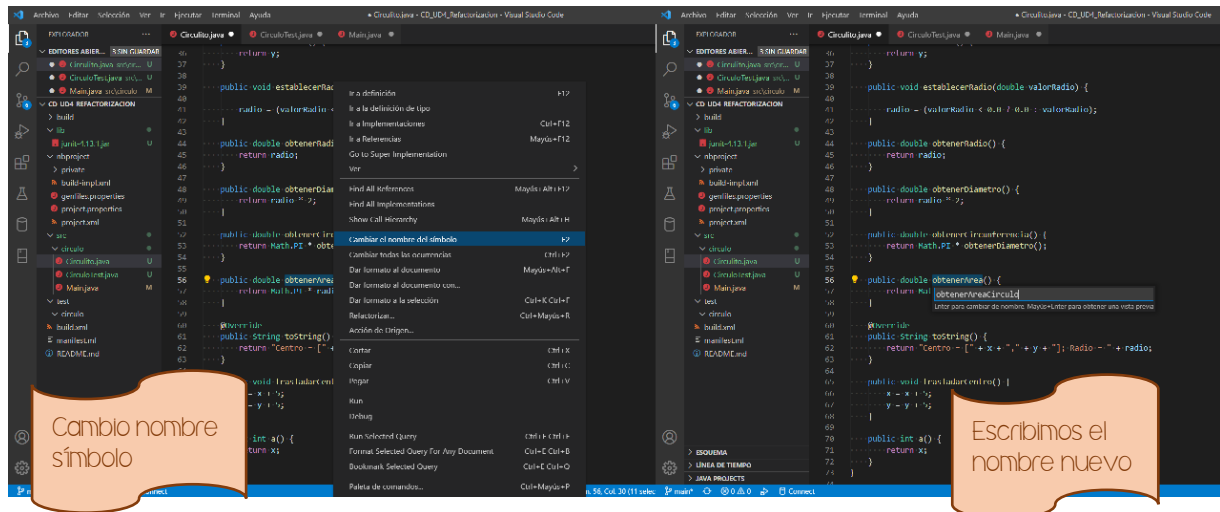


En la siguiente imagen, una vez aceptados los cambios, se puede ver tanto en la clase “Circulo” como en las clases “Main” y “CirculoTest”, se han realizado los cambios correctamente.



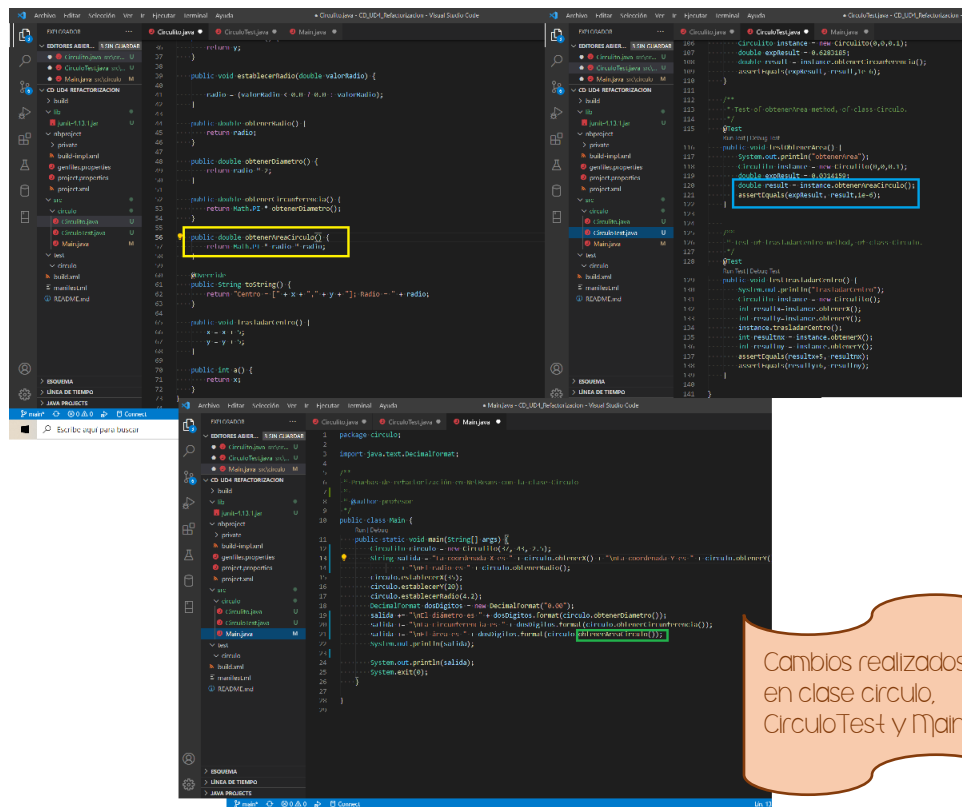
B. RENOMBRAR MÉTODO OBTENERAREA POR OBTENERAREACRICULO:

Para cambiar nombre al método obtenerArea por obtenerAreaCriculo, el procedimiento es el mismo que para la clase y el cambio se produce automáticamente en la clase "Circulo" como en las clases "CirculoTest" y "Main". Se muestran los pasos seguidos en las siguientes imágenes.



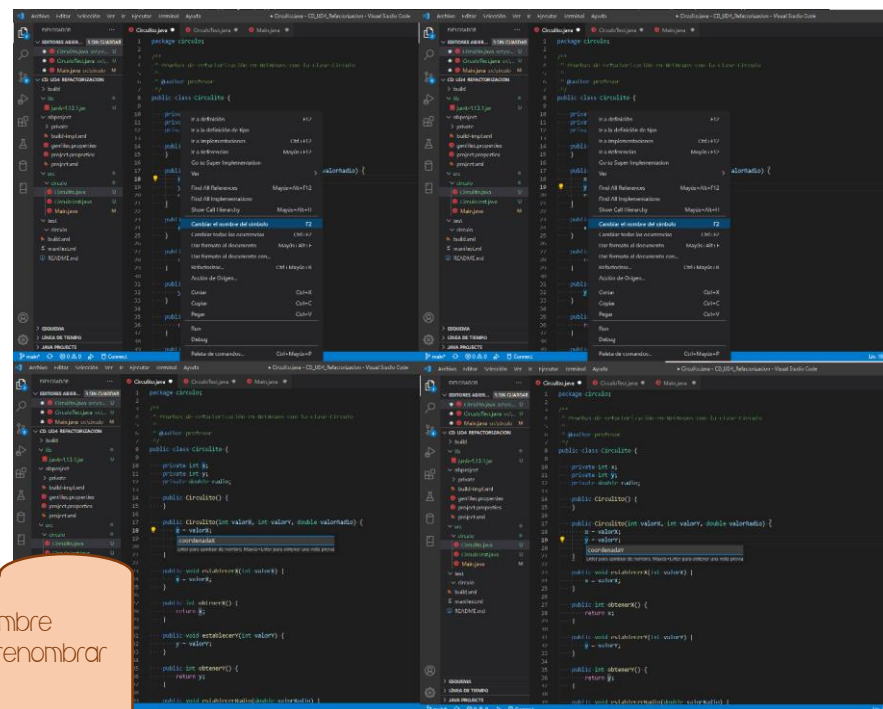
ED 4 – Tarea 4 Entornos de Desarrollo

Alumna: López Diéguez, Silvia

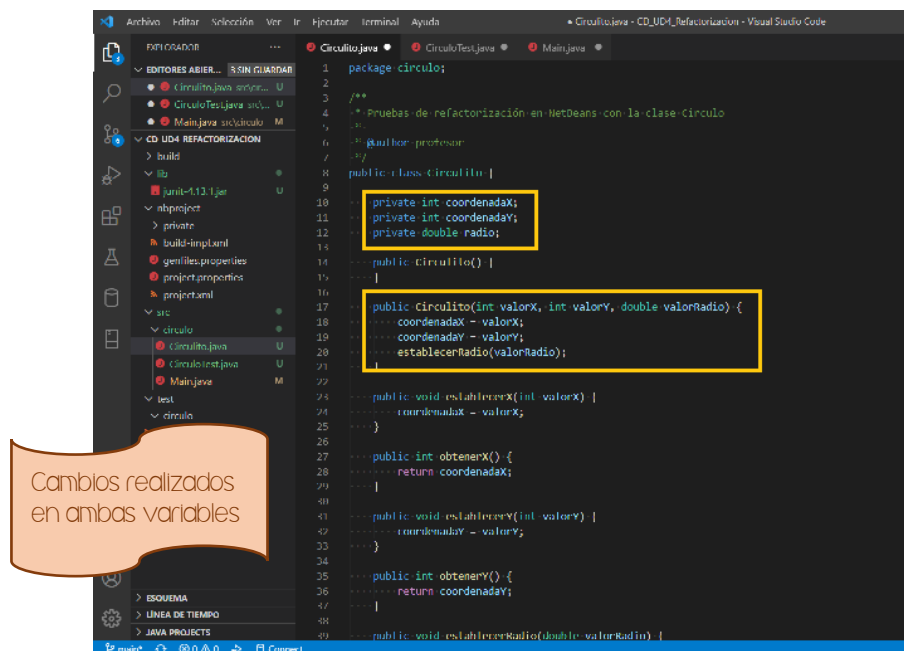
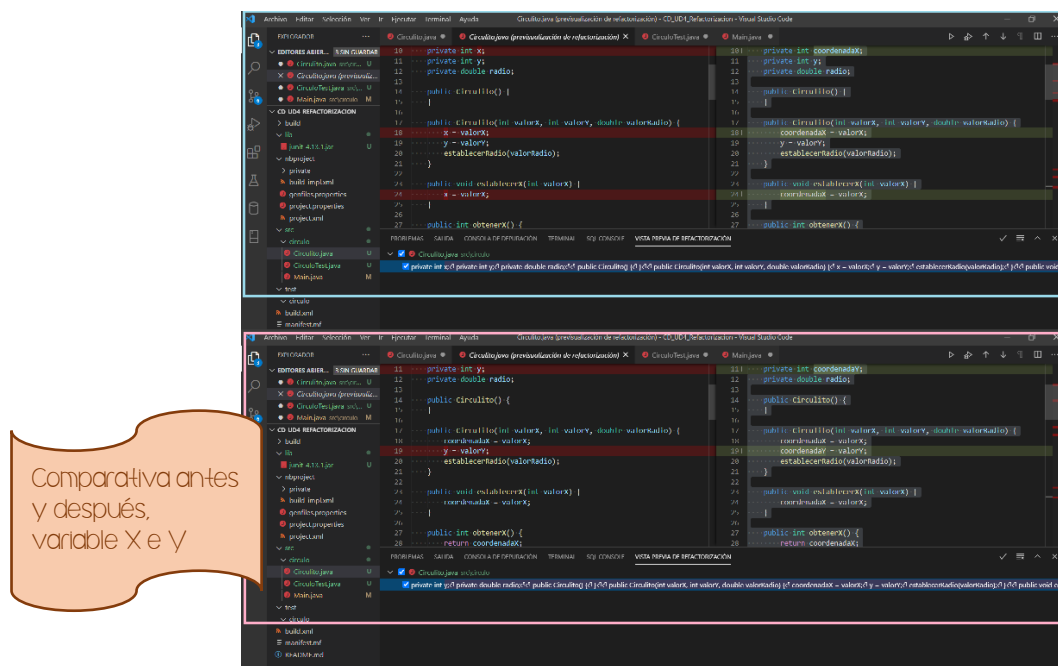


C. RENOMBRAR LOS CAMPOS X E Y POR COORDENADAX E COORDENADAY

Igual que en el cambio de nombre de la clase, y del método, para cambiar el nombre de las variables, el procedimiento es el mismo:



ED 4 – Tarea 4 Entornos de Desarrollo
Alumna: López Diéguez, Silvia



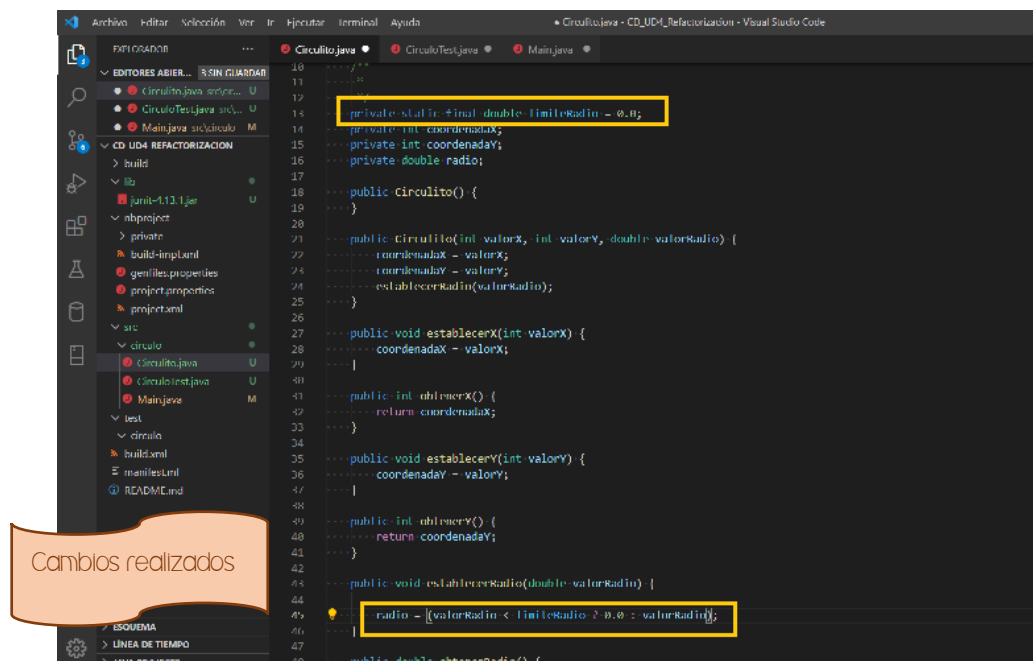
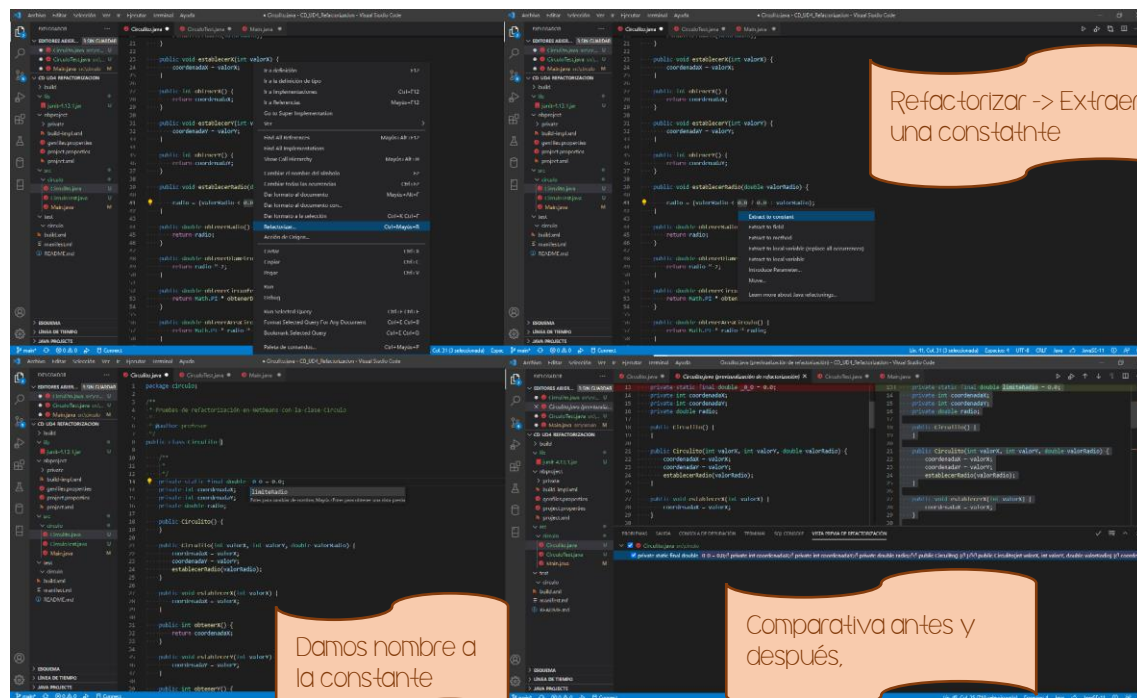
D. INTRODUCIR CONSTANTE LIMITERADIO DE TIPO DOUBLE CON EL VALOR 0.0

En este caso, no queremos que existan valores definidos a mitad del código, por lo que cambiamos el valor por una variable (en este caso una constante). Para ello, seleccionamos la opción Refactorizar, y seguidamente seleccionamos introducir una constante. El resultado será la definición de dicha constante junto con la definición del resto de variables, y en el código aparecerá el nombre de dicha constante en vez del valor.

ED 4 – Tarea 4 Entornos de Desarrollo

Alumna: López Diéguez, Silvia

En las siguientes imágenes se pueden ver los pasos seguidos en el cambio:



E. ELIMINAR DE FORMA SEGURA LOS MÉTODOS OBTENERX, OBTENERY, OBTENERRADIO, ESTABLECERX, ESTABLECERY, ESTABLECERRADIO, PARA QUE SEAN SUSTITUIDOS POR LOS CORRESPONDIENTES MÉTODOS GET Y SET CREADOS

Para eliminar de forma segura los métodos establecerX y obtenerX y sustituirlos por setCoordenadaX y getCoordenadaX seguimos los siguientes pasos. Clickamos con el botón derecho sobre “establecerX” y seleccionamos Refactorizar, seguido de “Inline Method”. Repetimos este paso para los demás métodos (obtenerX, establecerY, obtenerY, establecerRadio y obtenerRadio). De esta forma eliminamos los seis métodos.

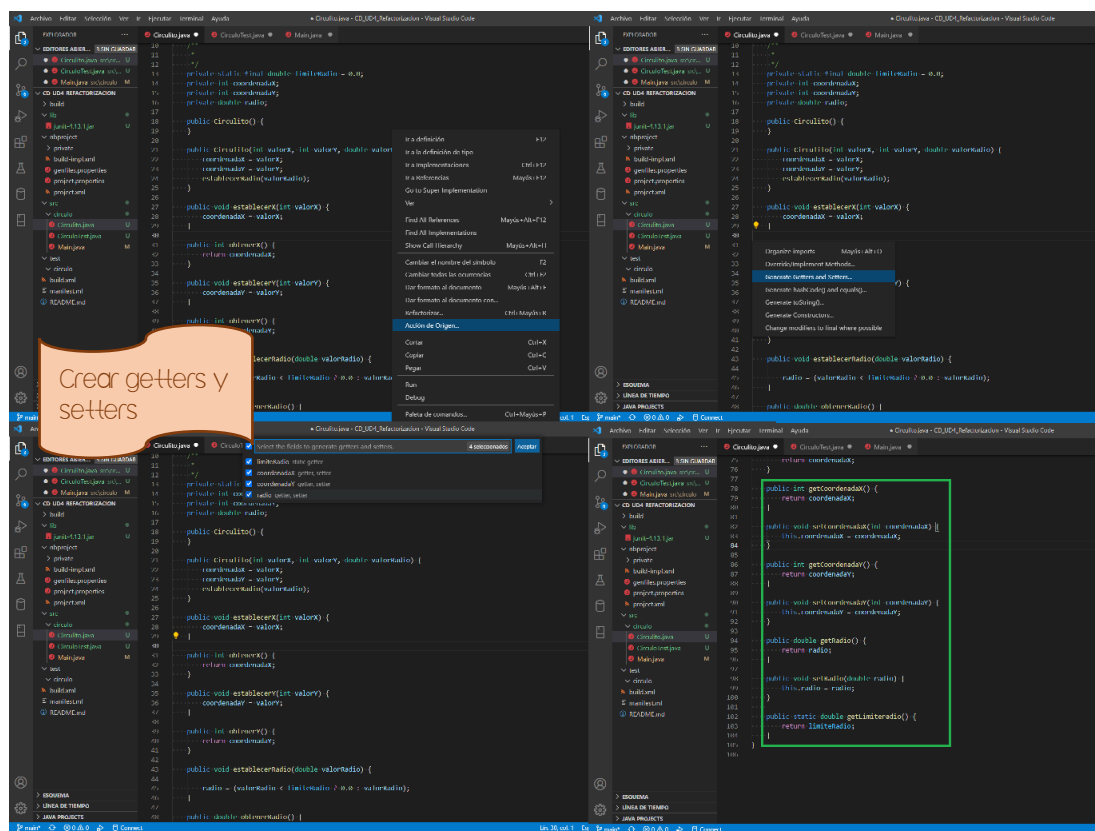


ED 4 – Tarea 4 Entornos de Desarrollo

Alumna: López Diéguez, Silvia

Una vez eliminados los métodos, los sustituimos por los getter y setter de esas variables. Para ello, clicamos con el botón derecho en el código y seleccionamos la opción “Acción de origen”, seguido de “Generate getters and setters”, y en el desplegable seleccionamos las variables sobre las que queremos crear los métodos.

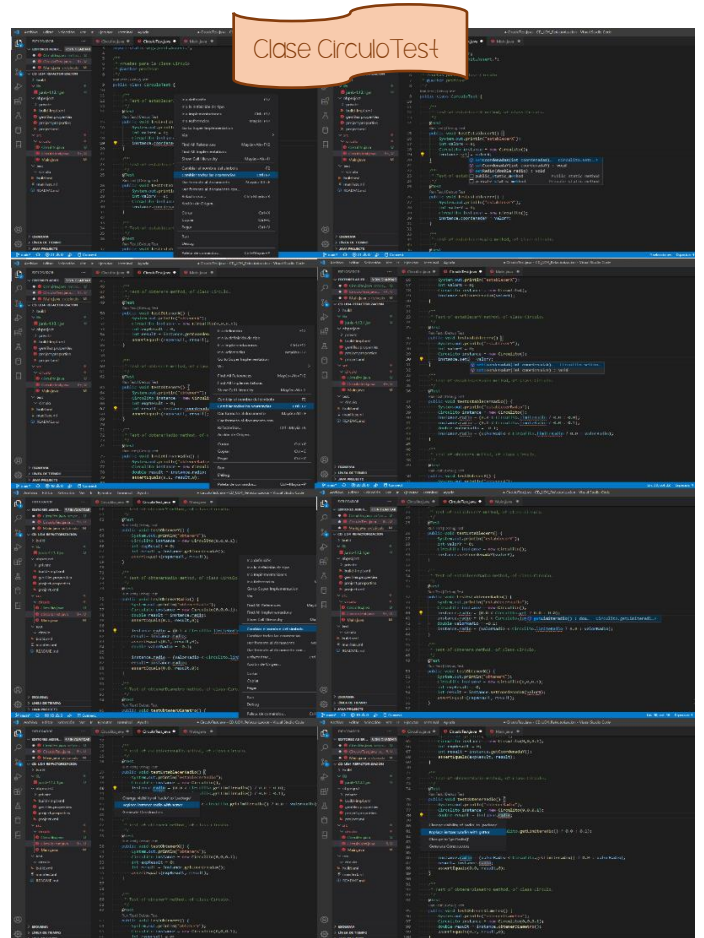
En el caso del método establecerRadio estaba definida la ecuación para calcular el radio. Al eliminar este método y crear el setter, cuando la clase main quiere obtener el valor de limiteRadio (el cual definimos como una constante), no tiene cómo hacerlo, por lo que es necesario crear su getter, para que al llamar a la clase Circulito, se pueda obtener el valor y realizar la ecuación



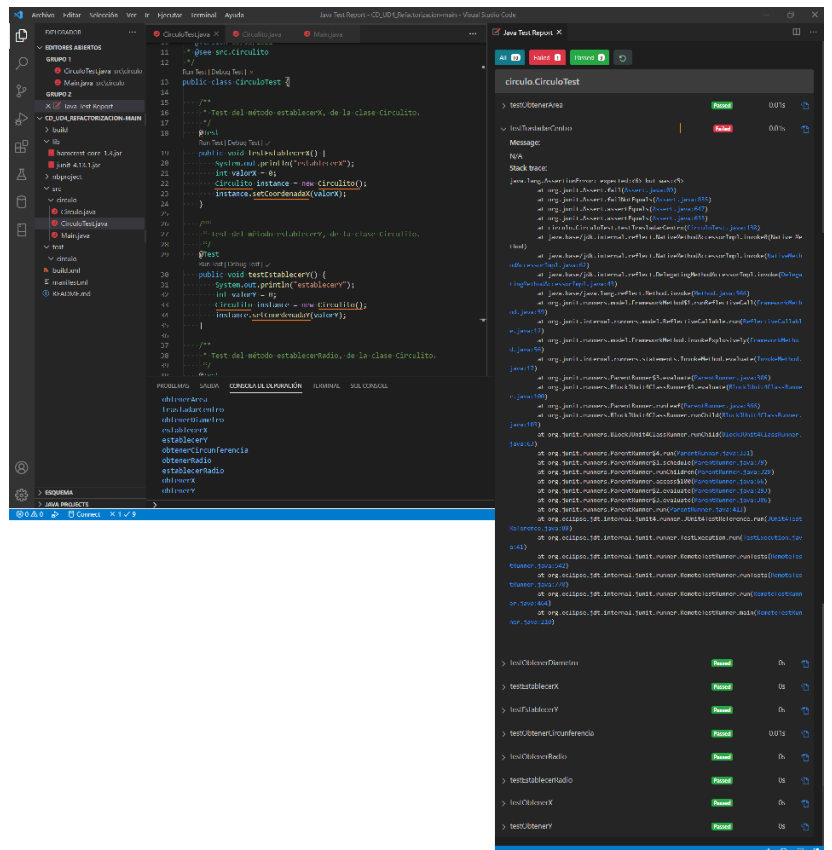
Como no se eliminan automáticamente en la clase main, accedemos a esta clase y sustituimos los anteriores métodos “establecer y obtener” por los recién creados métodos getters y setters correspondientes.

Sucede lo mismo con la clase CirculoTest. Como no se eliminan directamente, accedemos a esta clase y sustituimos los anteriores métodos “establecer y obtener” por los recién creados métodos getters y setters correspondiente. En esta clase, se hace cambiando el nombre, para que se cambie automáticamente en todas las instancias de la clase:

ED 4 – Tarea 4 Entornos de Desarrollo
Alumna: López Diéguez, Silvia



Una vez hechos los cambios mediante la refactorización, volvemos a compilar la aplicación para comprobar que el resultado es el mismo. Como se observa en la imagen de la derecha, los test que pasan la prueba unitaria son los mismos, igual que el test que no la pasa es el mismo test. Pues con la refactorización se busca mejorar el formato del código, para que sea más visual, más fácil de leer, no sea repetitivo, etc, pues de esa forma es más fácil de entender, pero no se busca corregir fallos relacionados directamente con el funcionamiento del código.



OPTATIVO: ENCAPSULAR LOS TRES CAMPOS DEL MÉTODO (COORDENADA X, COORDENADA Y, RADIO). INVESTIGAR LA FUNCIONALIDAD DE ENCAPSULAR

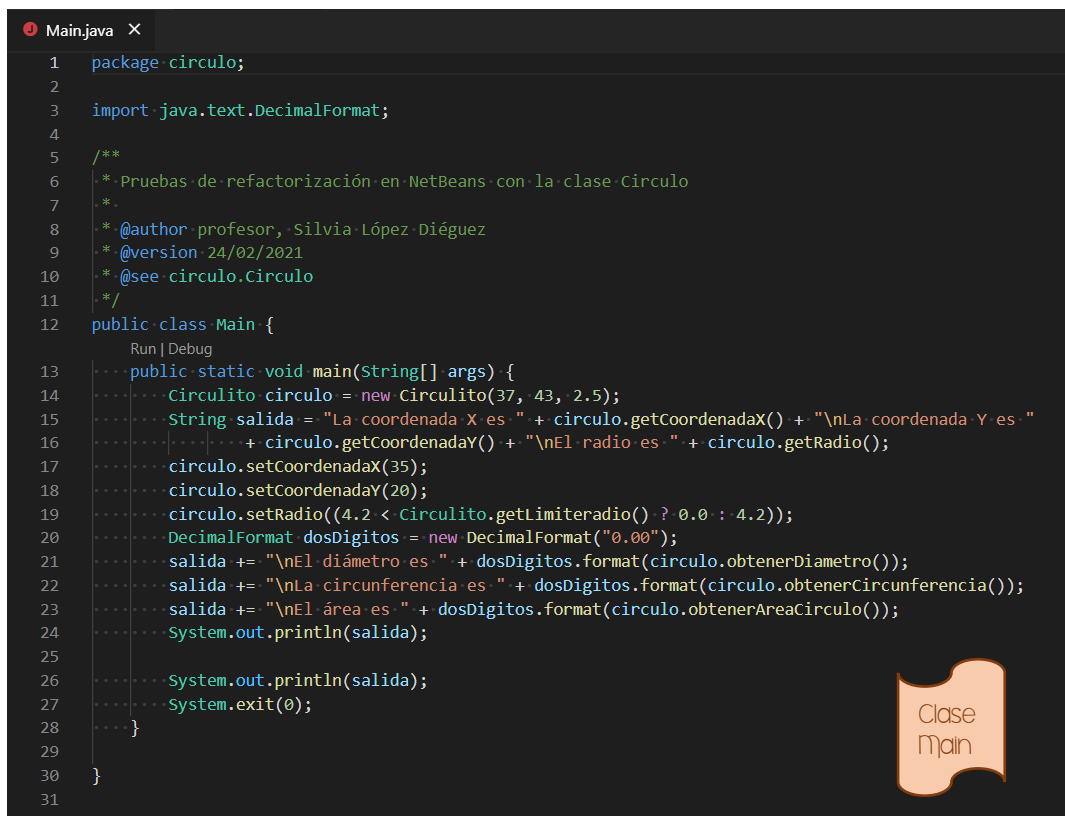
El encapsulamiento es limitar el acceso a las variables de una clase en Java, de manera que se pueda tener un mayor control sobre ellas y que no sean visibles ni se puedan modificar en el código desde otras clases. Para encapsular lo que se hace es privatizar los campos, es decir, establecer las variables como privadas, indicando `private` al iniciar cualquier variable. Posteriormente, para poder acceder a estas variables desde otras clases se crean los métodos `getter` y `setter` de cada variable. Con el método `setter` se puede dar valor a una variable, y con el método `getter` se accede a dicho valor.

En el caso de la clase `Circulito`, las variables ya están encapsuladas (cada variable va precedida por la palabra `private`).

Al sustituir los métodos “establecer” y “obtener” de cada variable, por los métodos `getter` y `setter`, terminamos de encapsular la clase, ya que con ellos permitimos el acceso a las variables desde otras clases sin comprometer su estado inicial. En este caso, desde la clase `Main`, gracias estos dos métodos, podemos dar un valor a cada variable y posteriormente acceder a dicho valor, desde la clase `Main`. (para lo cual tendríamos que hacerlo desde la clase en la que se encuentran, es decir, `Circulito`)

2. JAVADOC.

En las siguientes imágenes se muestran los comentarios realizados sobre el código para su posterior documentación con Javadoc, tanto de la clase `Circulito` como de la clase `Main`.



```
1 package circulo;
2
3 import java.text.DecimalFormat;
4
5 /**
6  * Pruebas de refactorización en NetBeans con la clase Circulo
7  *
8  * @author profesor, Silvia López Diéguez
9  * @version 24/02/2021
10 * @see circulo.Circulo
11 */
12 public class Main {
13     Run | Debug
14     public static void main(String[] args) {
15         Circulito circulo = new Circulito(37, 43, 2.5);
16         String salida = "La coordenada X es " + circulo.getCoordenadaX() + "\nLa coordenada Y es "
17             + circulo.getCoordenadaY() + "\nEl radio es " + circulo.getRadio();
18         circulo.setCoordenadaX(35);
19         circulo.setCoordenadaY(20);
20         circulo.setRadio((4.2 < Circulito.getLimiteradio() ? 0.0 : 4.2));
21         DecimalFormat dosDigitos = new DecimalFormat("0.00");
22         salida += "\nEl diámetro es " + dosDigitos.format(circulo.obtenerDiametro());
23         salida += "\nLa circunferencia es " + dosDigitos.format(circulo.obtenerCircunferencia());
24         salida += "\nEl área es " + dosDigitos.format(circulo.obtenerAreaCirculo());
25         System.out.println(salida);
26         System.out.println(salida);
27         System.exit(0);
28     }
29 }
30
31
```

Clase Main

ED 4 – Tarea 4 Entornos de Desarrollo
Alumna: López Diéguez, Silvia

```
Circulo.java intxano
package circulo;

/**
 * Pruebas de refactorización en NetBeans con la clase Circulo
 *
 * @author profesor, Silvia López Diéguez
 * @version 24/02/2021
 */
public class Circulo {
    // Campos de la clase Circulo
    private static final double limiteRadio = 0.0;
    private int coordenadaX;
    private int coordenadaY;
    private double radio;

    /**
     * Constructor vacío no asignación de argumentos de entrada
     */
    public Circulo() {
    }

    /**
     * Constructor para las coordenadas del círculo
     *
     * @param valorX ..... coordenada: coordenadaX
     * @param valorY ..... coordenada: coordenadaY
     * @param (valorRadio < limiteRadio ? 0.0 : valorRadio) ecuación: radio
     */
    public Circulo(int valorX, int valorY, double valorRadio) {
        coordenadaX = valorX;
        coordenadaY = valorY;
        radio = (valorRadio < limiteRadio ? 0.0 : valorRadio);
    }

    /**
     * Devuelve diámetro de círculo
     *
     * @return diámetro
     */
    public double obtenerDiámetro() {
        return radio * 2;
    }

    /**
     * Devuelve perímetro de círculo (circunferencia) y utiliza el valor de
     * PI-{@value java.lang.Math#PI}
     *
     * @return longitud
     */
    public double obtenerCircunferencia() {
        return Math.PI * obtenerDiámetro();
    }

    /**
     * Devuelve área de círculo y utiliza el valor de PI-{@value java.lang.Math#PI}
     *
     * @return área
     */
    public double obtenerAreaCirculo() {
        return Math.PI * radio * radio;
    }

    /**
     * Devuelve un resumen de todos los valores asignados a los campos
     *
     * @return valores de argumentos
     */
    @Override
    public String toString() {
        return "Centro = [" + coordenadaX + "," + coordenadaY + "]; Radio = " + radio;
    }

    /**
     * Desplaza el centro a una distancia 5
     *
     * @param coordenadaX suma 5 al valor de coordenadaX
     * @param coordenadaY suma 5 al valor de coordenadaY
     */
    public void trasladarCentro() {
        coordenadaX = coordenadaX + 5;
        coordenadaY = coordenadaY + 5;
    }

    /**
     * Devuelve el valor de un punto de eje X
     *
     * @return coordenadaX
     */
    public int a() {
        return coordenadaX;
    }

    /**
     * Devuelve la coordenadaX
     *
     * @return coordenadaX
     */
    public int getCoordenadaX() {
        return coordenadaX;
    }

    /**
     * Establece la coordenadaX
     *
     * @param coordenadaX
     */
    public void setCoordenadaX(int coordenadaX) {
        this.coordenadaX = coordenadaX;
    }

    /**
     * Establece la coordenadaY
     *
     * @param coordenadaY
     */
    public void setCoordenadaY(int coordenadaY) {
        this.coordenadaY = coordenadaY;
    }

    /**
     * Devuelve el valor del radio
     *
     * @return radio
     */
    public double getRadio() {
        return radio;
    }

    /**
     * Establece el valor del radio
     *
     * @param radio
     */
    public void setRadio(double radio) {
        this.radio = radio;
    }

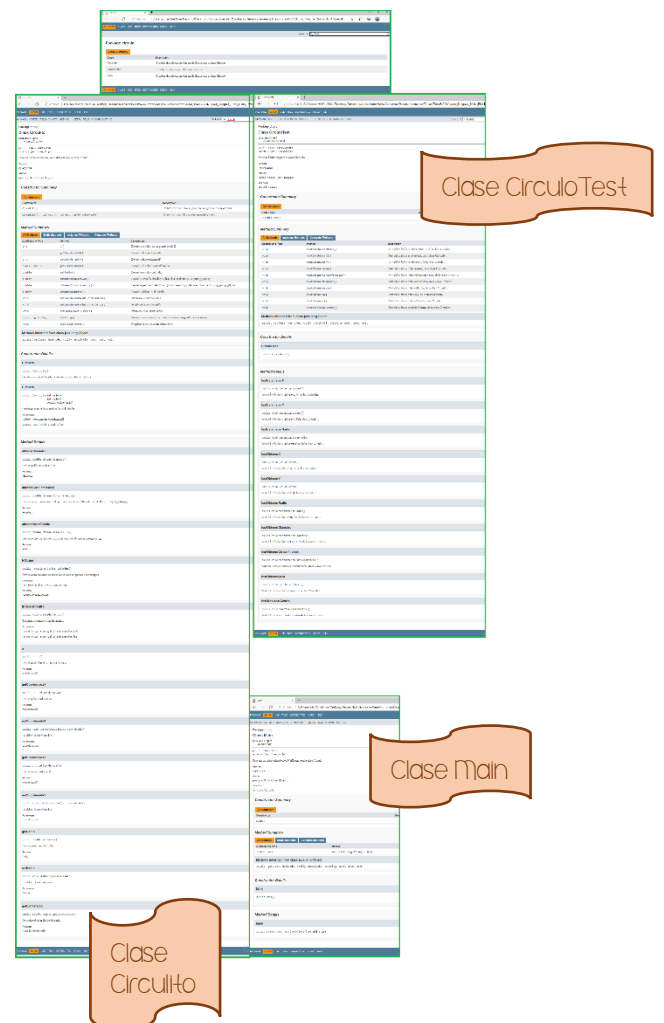
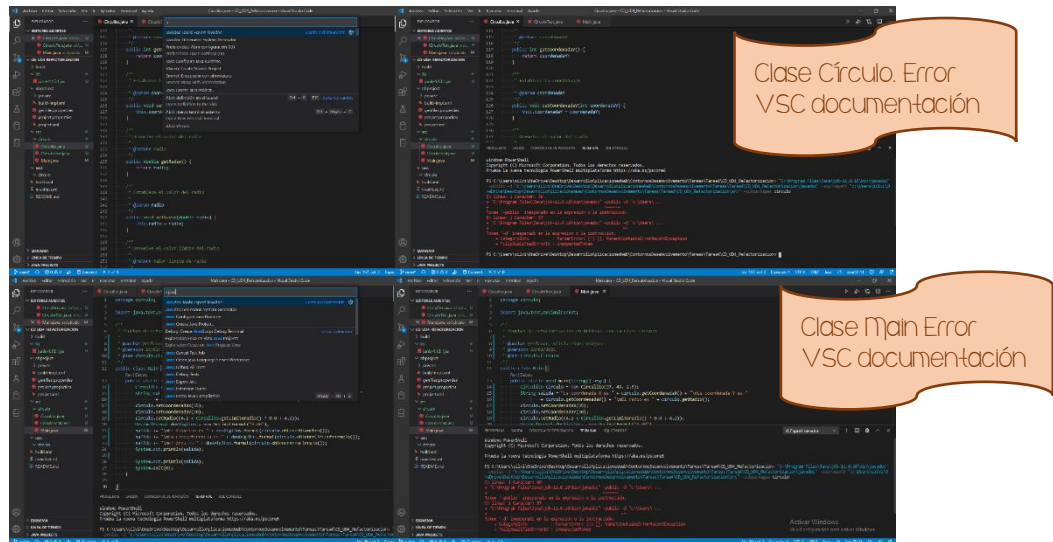
    /**
     * Devuelve el valor límite del radio
     *
     * @return valor límite de radio
     */
    public static double getLimiteRadio() {
        return limiteRadio;
    }
}
```

Clase
Circulo

ED 4 – Tarea 4 Entornos de Desarrollo

Alumna: López Diéguez, Silvia

Una vez introducimos etiquetas y comentarios en nuestro código, creamos el documento javadoc. Para ello, en la paleta de comandos de Visual Studio Code, escribimos la extensión Javadoc Tools:Export Javadoc que será la que nos generará la página con las instrucciones de nuestro código. En este caso, VSC nos da error al intentar generar la documentación, en las siguientes imágenes se ve el error:



En este caso, para comprobar resultados y ver cómo se genera o cuál es el resultado de las etiquetas introducidas en el proyecto, se importa el proyecto Selenium a un IDE diferente (en este caso Netbeans), y se obtiene la documentación, como se observa en la siguiente imagen:

3. SONARQUBE

En la siguiente imagen se muestra la creación de un proyecto Maven en Visual Studio Code: “pruebaSonarSilvia”, donde incluiremos las clases Circulito y Main para analizar el código con SonarQube.

```
PROBLEMAS  SALIDA  CONSOLA DE DEPURACIÓN  TERMINAL

[INFO] Parameter: version, value: 1.0
[INFO] Parameter: package, value: pruebaSonarSilvia
[INFO] Parameter: packageInPathFormat, value: pruebaSonarSilvia
[INFO] Parameter: package, value: pruebaSonarSilvia
[INFO] Parameter: groupId, value: pruebaSonarSilvia
[INFO] Parameter: artifactId, value: pruebaSonarSilvia
[INFO] Parameter: version, value: 1.0
[INFO] Project created from Archetype in dir: C:\Users\silvi\OneDrive\Documentos\VisualStudioCode\pruebaSonarSilvia
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 39.856 s
[INFO] Finished at: 2021-02-26T09:31:53+01:00
[INFO] -----
PS C:\Users\silvi\OneDrive\Documentos\VisualStudioCode> []
```

El resultado del análisis con SonarQube nos muestra seis “code smells”, lo que indica que es aconsejable refactorizar el código en seis puntos diferentes:

Métrica	Valor	Estado
Insectos	0	Aprobado
Vulnerabilidades	0	Aprobado
Hotspots revisados	-	Aprobado
Huelo a código	4	Aprobado
Cobertura	0,0%	Problema
Duplicaciones	0,0%	Problema
Líneas	139	OK

Code Smells:

- Realice las acciones necesarias para solucionar el problema indicado por este comentario "FIXME".** (Hace 10 segundos, L 12)
 - ¿Por qué es esto un problema? Código Olor (Importante)
- Cambie el nombre de este nombre de constante para que coincida con la expresión regular "[AZ][A-Z0-9]*_[A-Z0-9]".** (Hace 10 segundos, L 10)
 - ¿Por qué es esto un problema? Código Olor (Crítico)
- Reemplace este uso de System.out o System.err por un registrador.** (Hace 10 segundos, L 22)
 - ¿Por qué es esto un problema? Código Olor (Importante)
- Reemplace este uso de System.out o System.err por un registrador.** (Hace 10 segundos, L 24)
 - ¿Por qué es esto un problema? Código Olor (Importante)

Se describen a continuación los cuatro “Code Smells” que refleja el análisis de nuestro código:

1- Realice las acciones necesarias para solucionar el problema por este comentario “FIXME”

Hay momentos durante el desarrollo del código en que sabemos que hay algo que mejorar en alguna línea o algo que cambiar, o sospecha de un posible error. Como hay veces en los que no se puede hacer el cambio en el momento, realizando un comentario con la palabra FIXME, se agrega un recordatorio de ese cambio pendiente en una lista de tareas.

Aun estando en la lista de tareas, hay veces que al desarrollador se le puede pasar por alto, por lo que se refleja en el resultado del análisis del código como “Code Smell” para recordar el cambio pendiente de realizar.

2- Cambie el nombre de constante para que coincida con la expresión regular '[AZ][a-z0-9]*([A-Z0-9]+)*\$'

Con este aviso, nos indica que sería recomendable cambiar el nombre de nuestra variable por una cadena de caracteres que puede estar formada por letras mayúsculas, o letras mayúsculas y números o algún otro símbolo.

3y4- Reemplace este uso de System.out o System.err por un regulador

Tanto este aviso, como el siguiente, nos recomienda para realizar las impresiones de código usar método logger en vez del clásico System.out.println, pues el método logger permite guardar los mensajes que aparecen por consola en un archivo externo, tanto en texto plano (habría que indicarlo, ya que por defecto se guarda en formato html). Con cada mensaje que se guarde con este método, se registra el nombre completo de la clase, el método, la fecha, la hora y si existe algún error de nivel grave, la línea del código en la que se formó el reporte.

4. O NOSO PROXECTO

Documentamos nuestro proyecto de Selenium, para lo cual introducimos comentarios descriptivos sobre lo que realiza cada parte del código. En las siguientes imágenes podemos ver las etiquetas y los comentarios utilizados para el proyecto Selenium, tanto en la clase TestCases como en la clase App.

ED 4 – Tarea 4 Entornos de Desarrollo
Alumna: López Diéguez, Silvia

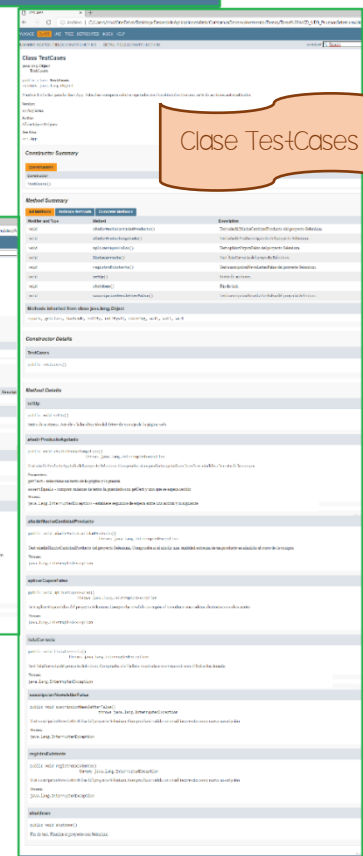
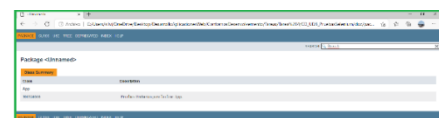
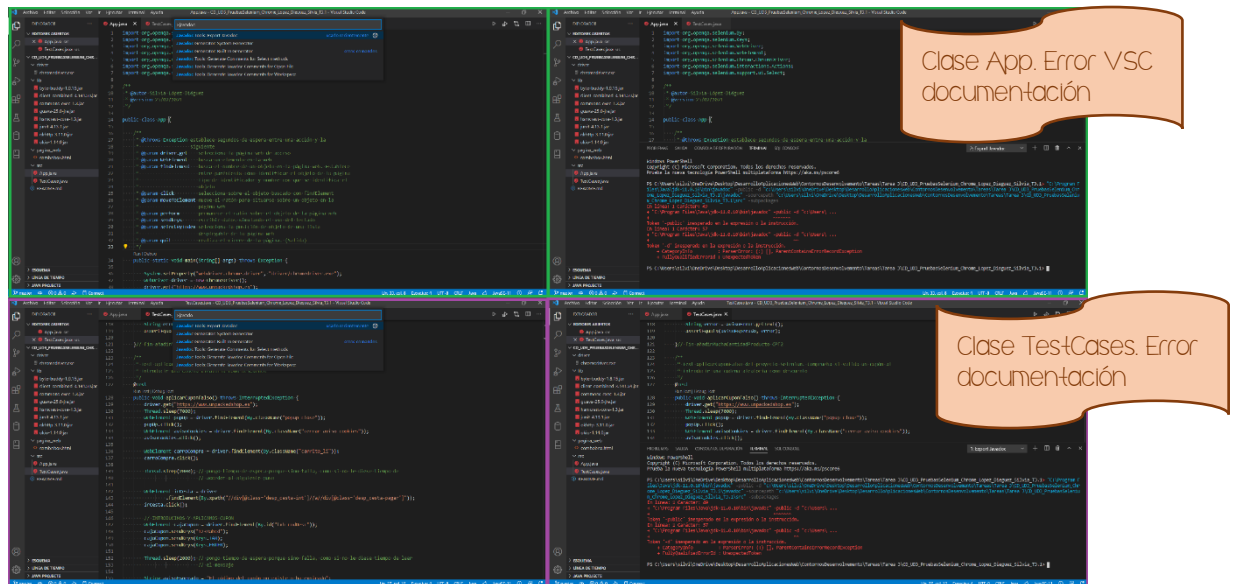
Clase
App

Clase
TestCases

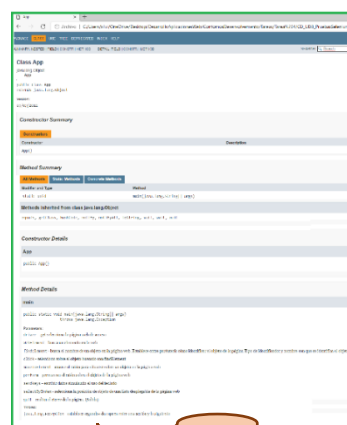
ED 4 – Tarea 4 Entornos de Desarrollo

Alumna: López Diéguez, Silvia

Una vez introducimos etiquetas y comentarios en nuestro código, creamos el documento javadoc. Para ello, en la paleta de comandos de Visual Studio Code, escribimos la extensión Javadoc Tools:Export Javadoc que será la que nos generará la página con las instrucciones de nuestro código. Al igual que en el proyecto Circulo, VSC nos da error al intentar generar la documentación, en las siguientes imágenes se ve el error.



Por lo que, para comprobar resultados y ver cómo se genera o cuál es el resultado de las etiquetas introducidas en el proyecto, se importa el proyecto Selenium a un IDE diferente (en este caso Netbeans), y se obtiene la documentación, como se observa en la siguiente imagen:



Clase App

Como en el proyecto de Circulo, creamos un análisis con SonarQube para el proyecto de Selenium. Una vez analizado el código, obtenemos los siguientes “Code Smells”

The screenshot displays the SonarQube web interface for a project named 'pruebaSeleniumSilvia'. The top navigation bar shows the project status as 'Aprobado' (Approved) and the last analysis was completed 25 seconds ago. Below this, a summary bar provides key metrics: 0 Insects, 0 Vulnerabilities, 0 Hotspots, 12 Code Smells, 0.0% Coverage, 0.0% Duplications, and 451 Lines of code. The main content area lists 12 code smells, each with a description, severity, and a 'FIXME' comment. The left sidebar contains filters for 'Código Olor' (Code Smell) and 'Gravedad' (Severity), with a 'Borrar todos los filtros' (Clear all filters) button. The code smells are categorized by severity: Critical (1), Major (5), and Minor (6).

Descripción	Gravedad	Comentario
Realice las acciones necesarias para solucionar el problema indicado por este comentario "FIXME".	Crítico	¿Por qué es esto un problema?
Mueva este archivo a un paquete con nombre.	Menor	¿Por qué es esto un problema?
Cambie el nombre de esta variable local para que coincida con la expresión regular "[a-z][a-zA-Z0-9]"	Menor	¿Por qué es esto un problema?
Defina una constante en lugar de duplicar este literal "Apoyo_Mensaje_div_aceptar" 3 veces.	Crítico	¿Por qué es esto un problema?
Mueva este archivo a un paquete con nombre.	Menor	¿Por qué es esto un problema?
Cambie el nombre de este método para que coincida con la expresión regular "[a-z][a-zA-Z0-9]"	Menor	¿Por qué es esto un problema?
Defina una constante en lugar de duplicar este literal "https://www.unpackedshop.es" 6 veces.	Crítico	¿Por qué es esto un problema?
Defina una constante en lugar de duplicar este literal "popup_close" 6 veces.	Crítico	¿Por qué es esto un problema?
Defina una constante en lugar de duplicar este literal "cerrar_aviso_cookies" 6 veces.	Crítico	¿Por qué es esto un problema?
Cambie el nombre de esta variable local para que coincida con la expresión regular "[a-z][a-zA-Z0-9]"	Menor	¿Por qué es esto un problema?
Cambie el nombre de este método para que coincida con la expresión regular "[a-z][a-zA-Z0-9]"	Menor	¿Por qué es esto un problema?
Defina una constante en lugar de duplicar este literal "Apoyo_Mensaje_div_texto" 4 veces.	Crítico	¿Por qué es esto un problema?

Se describen a continuación los doce “Code Smells” que refleja el análisis del código de las pruebas unitarias hechas con Selenium:

1- Realice las acciones necesarias para solucionar el problema por este comentario "FIXME"

Como se explicó para el código de Circulo, hay momentos durante el desarrollo del código en que sabemos que hay algo que mejorar en alguna línea o algo que cambiar, o sospecha de un posible error. Como hay veces en los que no se puede hacer el cambio en el momento, realizando un comentario con la palabra FIXME, se agrega un recordatorio de ese cambio pendiente en una lista de tareas.

Aun estando en la lista de tareas, hay veces que al desarrollador se le puede pasar por alto, por lo que se refleja en el resultado del análisis del código como “Code Smell” para recordar el cambio pendiente de realizar.

- 2- Mueva este archivo a un paquete con nombre (también el MENSAJE 5 de nuestro análisis)

La clase App.java no pertenece a ningún paquete, por lo que con este aviso nos está recomendando incluirlo en un paquete, siguiendo con las normas de modularidad que sigue el lenguaje Java .

Este mensaje se repite en **quinto** lugar para la clase TestCases.java por el mismo motivo.

- 3- Cambie el nombre de constante para que coincida con la expresión regular [AZ] [a-z0-9]*([A-Z0-9]+)*\$ (también para los MENSAJES 6, 10 y 11)

Igual que en el proyecto de Círculo, el nombre escogido para el objeto lo entiende como poco compatible o incompatible con la convención de nomenclatura que deben seguir los parámetros de métodos y variables locales del código.

En el **segundo** y el **décimo**, apunta al nombre “añadir”, y puede ser debido a que la que “ñ” sea interpretado como un carácter inadecuado, o que no reconozca dicho carácter.

Sucede lo mismo para los casos **sexto** y **décimo primero**, en los que se resaltan los nombres “añadirProductoAgotado” y “añadirMuchaCantidadProducto”, se puede entender que esta alerta la resalta por el carácter “ñ”, que puede que lo interprete como inadecuado o que no lo reconozca como una letra.

- 4- Defina una constante en lugar de duplicar este literal “Apoyo_Mensaje_div_aceptar” 3 veces (también el MENSAJE 7,8,9 y 12)

En esta alerta nos indica que a lo largo del código se repite el nombre de la variable de manera literal número elevado de veces. Nos aconseja, para que sea más fácil de leer el código e incluso más corto, que busquemos la manera de extraer y declarar esta variable como constante al inicio del código para que sea más fácil de seguir.

Para el **cuarto** caso, sucede con “Apoyo_Mensaje_div_aceptar”, que se repite tres veces a lo largo del código.

En los casos **séptimo**, **octavo**, **noveno** y **duodécimo** sucede (respectivamente) con <http://www.unpackedshop.es>, “popup_close”, “cerrar_aviso_cookies” y “Apoyo_Mensaje_div_aceptar”. Todos ellos se repiten 6 veces, y es debido a que forman parte de la clase TestCases.java, que es la clase que realiza las pruebas unitarias. Se trata de 6 test unitarios que comparten una parte del código, que, como nos indica “Case Smells” podrían extraerse como constantes para así hacer el código más corto.