

Entornos de desarrollo - Tarea 4

OPTIMIZACIÓN Y DOCUMENTACIÓN.

MIGUEL LLORENTE GONZÁLEZ

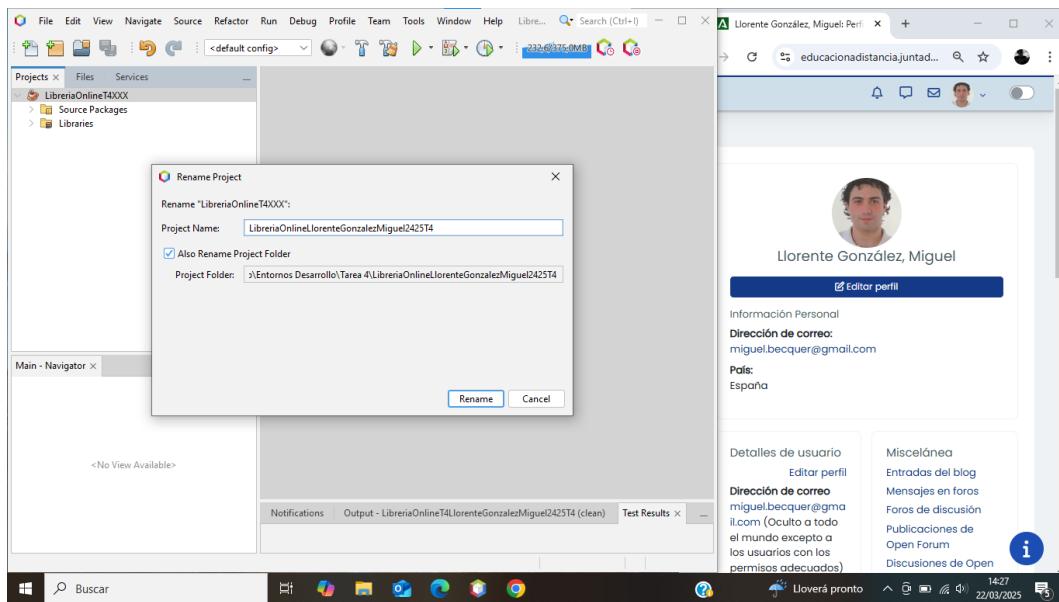
Curso Académico: 2024-2025

Utiliza las herramientas de refactorización del IDE NetBeans

1- Realizar toda la refactorización solicitada al principio como requisito indispensable para la corrección de la tarea (proyecto, clase y variable de la clase Main).

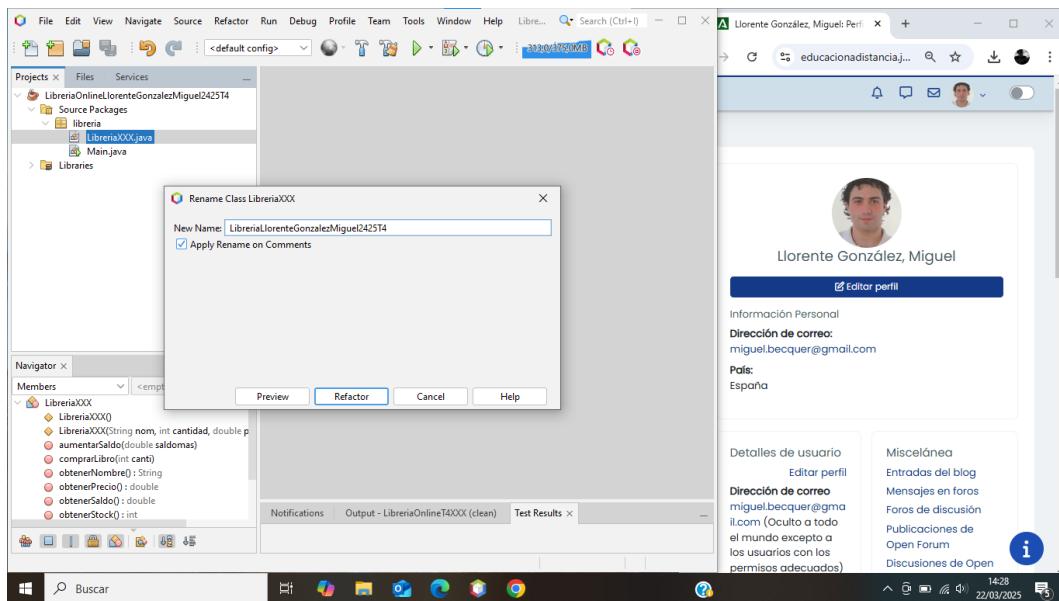
Descargamos el archivo del apartado ‘información de interés’, lo descomprimimos y, desde NetBeans, abrimos el proyecto.

Para cambiar el nombre del proyecto, hacemos clic derecho en él en el panel izquierdo y seleccionamos ‘Rename...’. Se abrirá una ventana donde podemos decidir el nuevo nombre, en este caso el indicado en el enunciado según mi nombre. También marco la casilla que indica que también se cambiará el nombre de la carpeta del proyecto:



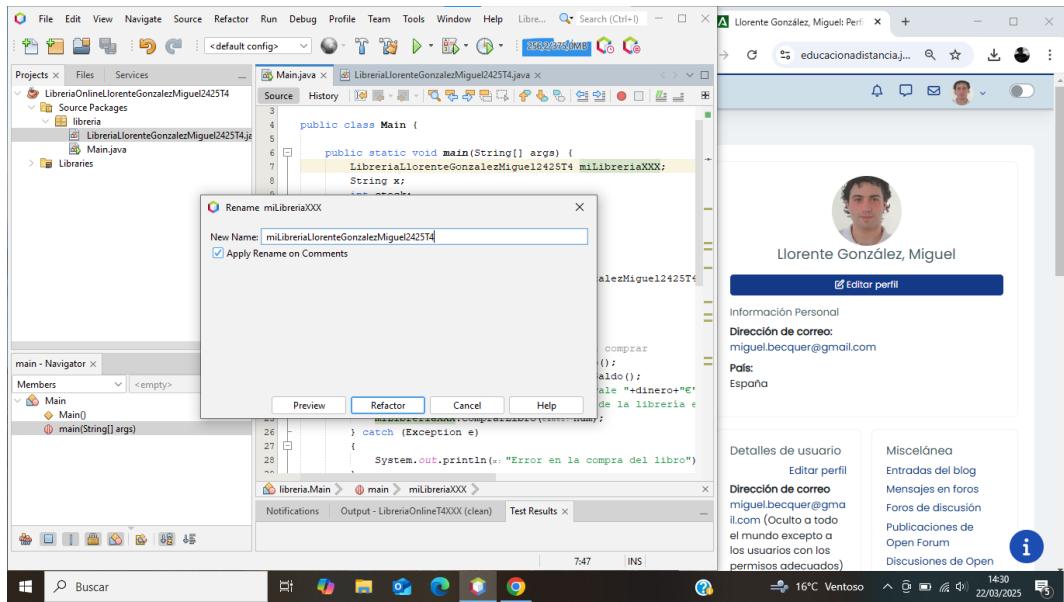
Hacemos clic en ‘Rename’.

Ahora, tenemos que cambiar el nombre de la clase ‘LibreriaXXX’. Para ello, en el mismo panel izquierdo, la seleccionamos haciendo clic derecho. Hacemos clic en ‘Refactor>Rename...’ y, en la ventana que aparece, introducimos el nuevo nombre:

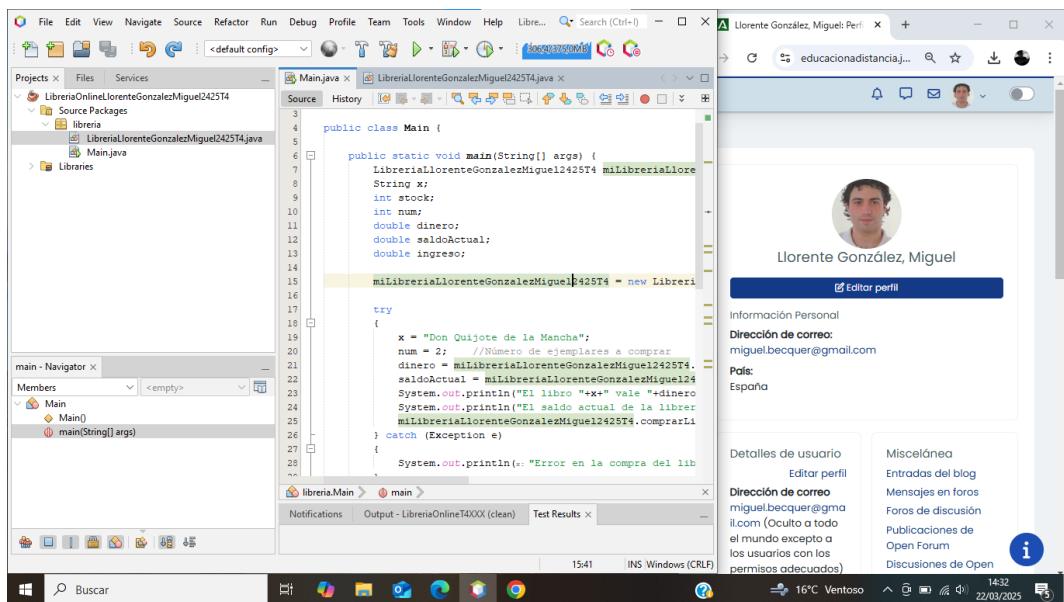


Hacemos clic en ‘Refactor’.

Por último, tenemos que cambiar el nombre de la variable ‘miLibreriaXXX’ que se encuentra en el código de la clase ‘Main’. La seleccionamos haciendo clic en ella, quedando resaltada en cada lugar del código en el que esté escrita. Hacemos clic derecho en ella (en cualquier lugar del código en el que se encuentre, no hace falta que sea en el lugar en el que se declara) y seleccionamos ‘Refactor>Rename...’. Escribimos el nuevo nombre y hacemos clic en ‘Refactor’:



En la siguiente captura podemos ver realizados correctamente los tres cambios mediante la refactorización del código:



2- Introduce el método `compraQuijoteXXX`, que englobe todas las sentencias de la clase `Main` necesarias para realizar la compra del libro de *Don Quijote de la Mancha*.

Vamos a introducir un método de compra del libro ‘*Don Quijote de la Mancha*’ que contenga las líneas de código en la clase ‘`Main`’ destinadas a ello. Estas líneas son de la 17 a la 29. Se trata de un try-catch que ejecutará el método ‘`comprarLibro`’ sobre una librería creada, cuyo nombre hemos refactorizado antes, que es objeto de la clase también refactorizada en el ejercicio anterior. Seleccionamos entonces las líneas de código que comprende el try-catch y hacemos clic derecho, escogiendo la opción ‘Refactor>Introduce>Method...’. Aparecerá una ventana en la que debemos escribir el nombre del método creado, que será el indicado en el enunciado. Hacemos clic en ‘Ok’ y tendremos el método al final del código de la clase, y el uso del método creado en lugar del código seleccionado anteriormente:

The screenshot shows the Eclipse IDE interface. On the left, the Project Explorer displays a Java project named 'LibreriaOnlineLlorenteGonzalez'. The 'Main.java' file is open in the editor. A 'Source' tab is selected. Lines 17 through 29 of the code are highlighted in blue, indicating they are selected for refactoring. A context menu is open over these lines, and the 'Introduce Method' option is being selected. A modal dialog titled 'Introduce Method' appears, showing the new method name 'compraQuijoteLlorenteGonzalezMiguel2425T4' and access level 'public'. The 'Target Class' dropdown is set to 'libreria.Main'. The 'Ok' button is highlighted.

```

16
17     {
18         String x = "Don Quijote de la Mancha";
19         num = 2; //Número de ejemplares a comprar
20         dinero = miLibreriaLlorenteGonzalezMiguel2425T4.obtenerPrecio();
21         saldoActual = miLibreriaLlorenteGonzalezMiguel2425T4.obtenerSaldo();
22         System.out.println("El libro "+x+" vale "+dinero+"€");
23         System.out.println("El saldo actual de la librería es de "+saldoActual);
24         miLibreriaLlorenteGonzalezMiguel2425T4.comprarLibro(saldo, num);
25     } catch (Exception e)
26     {
27         System.out.println("Error en la compra del libro");
28     }
29 }

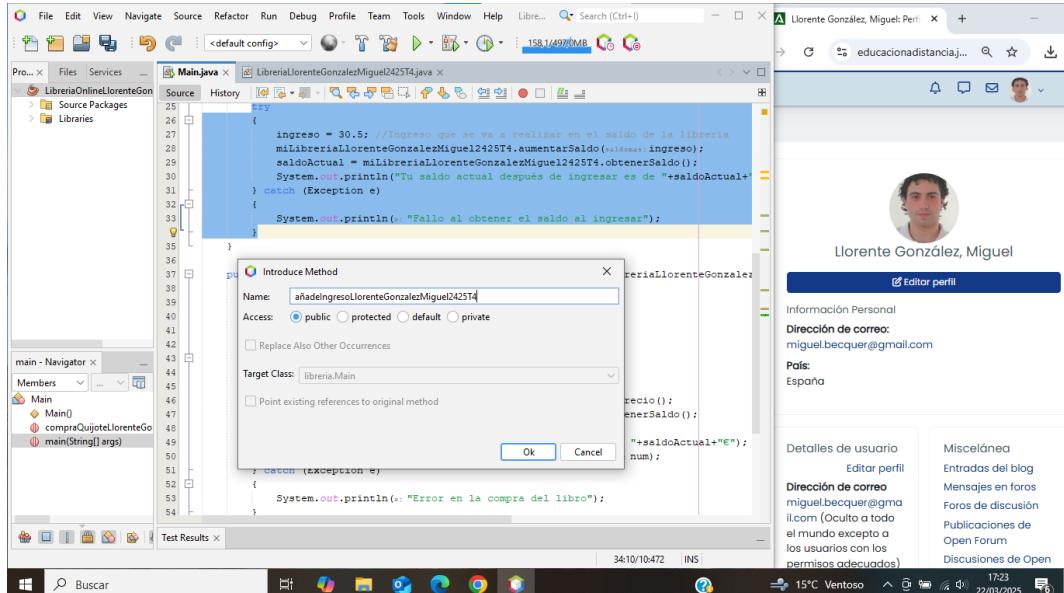
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57

```

The code in the editor has been modified to reflect the refactoring. The original code from line 17 to 29 has been replaced by a call to the new method 'compraQuijoteLlorenteGonzalezMiguel2425T4'. The method signature is identical to the original code it replaced.

3- Introduce el método `añadeIngresoXXX`, que englobe todas las sentencias de la clase Main necesarias para aumentar el saldo con el ingreso de la cantidad de dinero declarada.

Repetimos el proceso del apartado 2, esta vez para las líneas de código 26 a 34 que consisten en un try-catch que, haciendo uso del método existente ‘aumentarSaldo’, realizarán un ingreso de 30.5€ en el objeto ‘miLibreriaLlorenteGonzalezMiguel2425T4’. De esta manera, se crea un método que realizará un ingreso de dicha cantidad al objeto de la clase que se indique:



The screenshot shows the Eclipse IDE interface with the Main.java file open. A tooltip 'Introduce Method' is displayed over the code. The 'Name:' field contains 'añadeIngresoLlorenteGonzalezMiguel2425T4'. The 'Access:' dropdown is set to 'public'. The 'Target Class:' dropdown is set to 'libreria.Main'. The 'Point existing references to original method' checkbox is unchecked. The code in the tooltip is:

```

    Name: añadeIngresoLlorenteGonzalezMiguel2425T4
    Access: public
    Target Class: libreria.Main
    Point existing references to original method: false
    
```

The main code in Main.java is as follows:

```

    25
    26     {
    27         ingreso = 30.5; //Ingreso que se va a realizar en el saldo de la libreria
    28         miLibreriaLlorenteGonzalezMiguel2425T4.aumentarSaldo(saldoActual, ingreso);
    29         saldoActual = miLibreriaLlorenteGonzalezMiguel2425T4.obtenerSaldo();
    30         System.out.println("Tu saldo actual después de ingresar es de "+saldoActual);
    31     } catch (Exception e)
    32     {
    33         System.out.println("Fallo al obtener el saldo al ingresar");
    34     }
    35 }
    36
    37 public static void añadeIngresoLlorenteGonzalezMiguel2425T4()
    38 {
    39     double ingreso;
    40     double saldoActual;
    41     try
    42     {
    43         ingreso = 30.5; //Ingreso que se va a realizar en el saldo de la libreria
    44         miLibreriaLlorenteGonzalezMiguel2425T4.aumentarSaldo(saldoActual, ingreso);
    45         saldoActual = miLibreriaLlorenteGonzalezMiguel2425T4.obtenerSaldo();
    46         System.out.println("Tu saldo actual después de ingresar es de "+saldoActual);
    47     } catch (Exception e)
    48     {
    49         System.out.println("Fallo al obtener el saldo al ingresar");
    50     }
    51 }
    52
    53 public static void compraQuijoteLlorenteGonzalezMiguel2425T4(LibreriaLlorenteGonzalezMiguel2425T4 libro)
    54 {
    55     String x;
    56     int num;
    57     double dinero;
    58     double saldoActual;
    59     try
    60     {
    61         libro.compraQuijote();
    62         libro.añadeIngreso();
    63     } catch (Exception e)
    64     {
    65         System.out.println("Fallo al comprar el libro");
    66     }
    67 }
    68
    69 public static void main(String[] args)
    70 {
    71     Main.main0();
    72 }
    73
    74 
```



The profile page displays the following information:

- Información Personal:** Dirección de correo: miguel.becquer@gmail.com, País: España
- Miscelánea:** Entradas del blog, Mensajes en foros, Foros de discusión, Publicaciones de Open Forum, Discusiones de Open

4- Encapsula todos los atributos de la clase LibreriaXXX.

Nos situamos en la clase. Hacemos clic derecho en cualquier parte del código y seleccionamos ‘Refactor>Encapsulate Fields...’. En la ventana que aparece, hacemos clic en ‘Select all’ y luego en ‘Refactor’. Se pueden ver los métodos creados:

The screenshot shows two instances of the Eclipse IDE. The top instance displays the 'Encapsulate Fields' dialog for the file 'Main.java'. The 'Field' section lists four fields: 'nom: String', 'cantidad: int', 'precio: double', and 'saldo: double'. The 'Create Getter' and 'Create Setter' checkboxes are checked for all fields. The 'Insert Point' dropdown is set to 'Default'. Below the dialog, the code editor shows the generated getters and setters. The bottom instance shows the resulting code in 'Main.java' with the new methods added.

```

    nom: String
    cantidad: int
    precio: double
    saldo: double

    /**
     * @return the nom
     */
    public String getNom() {
        return nom;
    }

    /**
     * @param nom the nom to set
     */
    public void setNom(String nom) {
        this.nom = nom;
    }

    /**
     * @return the cantidad
     */
    public int getCantidad() {
        return cantidad;
    }

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

    /**
     * @return the precio
     */
    public double getPrecio() {
        return precio;
    }
}

```

5- Añadir un nuevo parámetro al método `añadeIngresoXXX` de la clase `LibreriaXXX`, de nombre *concepto*, de tipo *String* y valor por defecto ‘Libro vendido’.

Nos dirigimos al método creado que, al contrario de lo que se indica en el enunciado, se encuentra en la clase Main. Seleccionamos el método y hacemos clic derecho sobre él. Seleccionamos ‘Refactor>Change Method Parameters...’. Se abre una ventana en la que tenemos el parámetro que se usa. Hacemos clic en ‘Add’ para añadir uno nuevo. Introducimos los datos correspondientes y hacemos clic en ‘Refactor’. Ahora podemos ver que se ha incluido el parámetro, tanto en la declaración del método como en su uso:

The screenshot shows the Eclipse IDE interface with two windows open. On the left, the 'Main.java' file is displayed in the editor, showing the modified code with the new parameter 'concepto'. On the right, a user profile window for 'Llorente González, Miguel' is open, showing the user's information and activity feed. The code in the editor is as follows:

```

public static void añadeIngreso(LibreriaLlorenteGonzalezMiguel2425T4 miLibreriaLlorenteGonzalezMiguel2425T4, String concepto) {
    miLibreriaLlorenteGonzalezMiguel2425T4.setStock();
    miLibreriaLlorenteGonzalezMiguel2425T4.setStock(miLibreriaLlorenteGonzalezMiguel2425T4.getStock() + concepto);
}

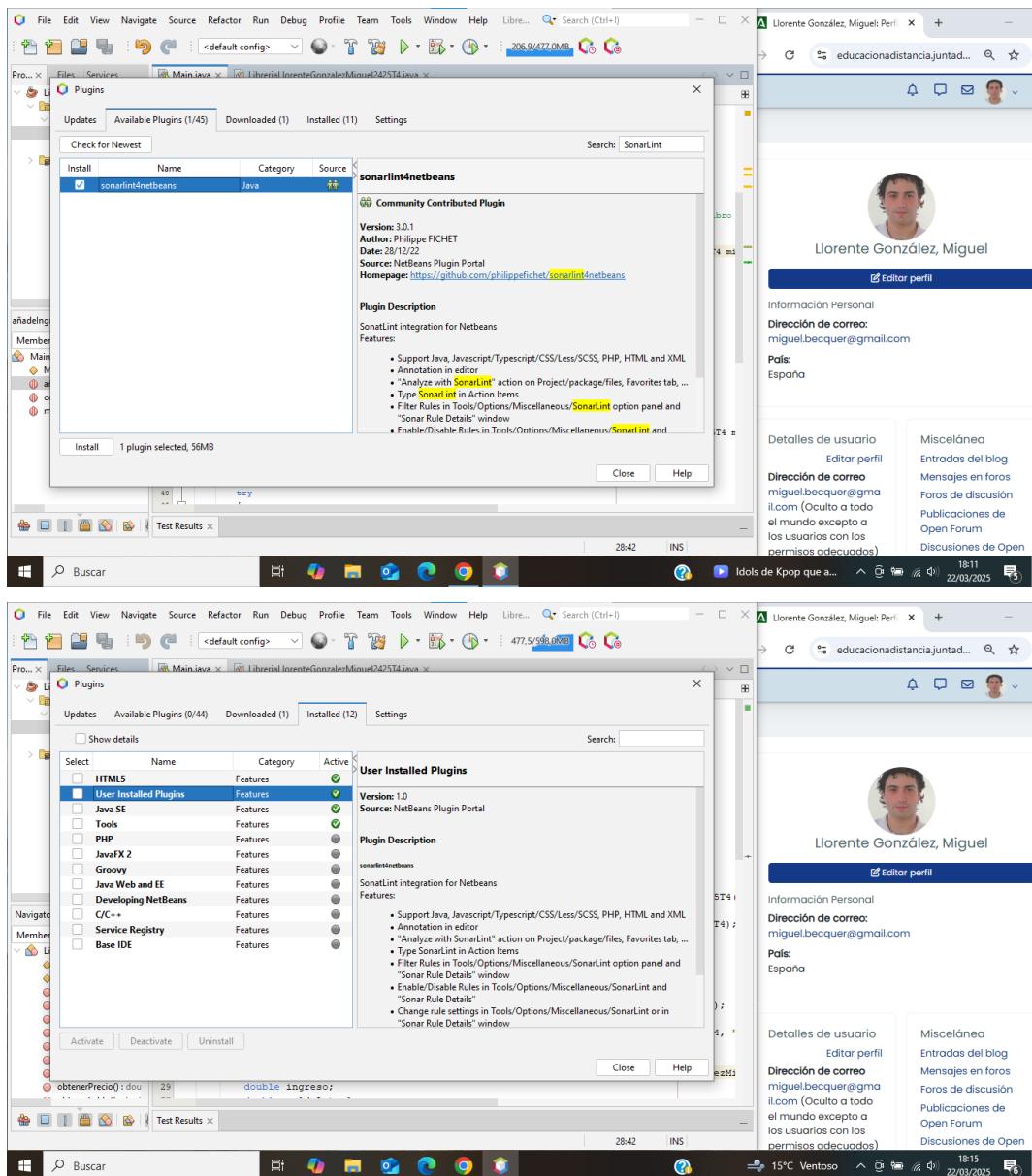
```

The 'Change Method Parameters' dialog is also visible, showing the new parameter 'concepto' added to the method signature.

Realiza un análisis de código.

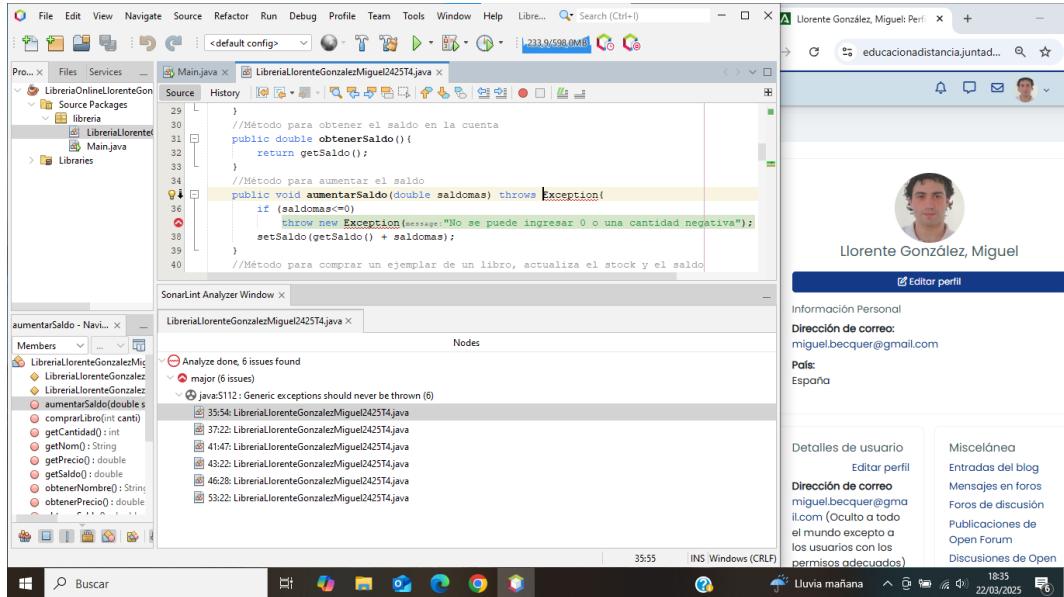
6- Instala el plug-in SonarLint en NetBeans (del autor FICHET Philippe).

Hacemos clic en ‘Tools’ (herramientas) en el panel superior y escogemos ‘Plugins’. En la ventana que aparece, tendremos que situarnos en la pestaña ‘Available Plugins’ (Plugins Disponibles) e introducimos, en el campo de búsqueda, ‘SonarLint’. Aparece en el panel izquierdo. Lo seleccionamos y hacemos clic en ‘Install’ (Instalar). Se abre una ventana con el instalador, donde completaremos la instalación. Se reinicia el IDE NetBeans y está el Plugin instalado, pudiéndolo encontrar en la pestaña ‘Installed’ de la ventana ‘Plugins’:

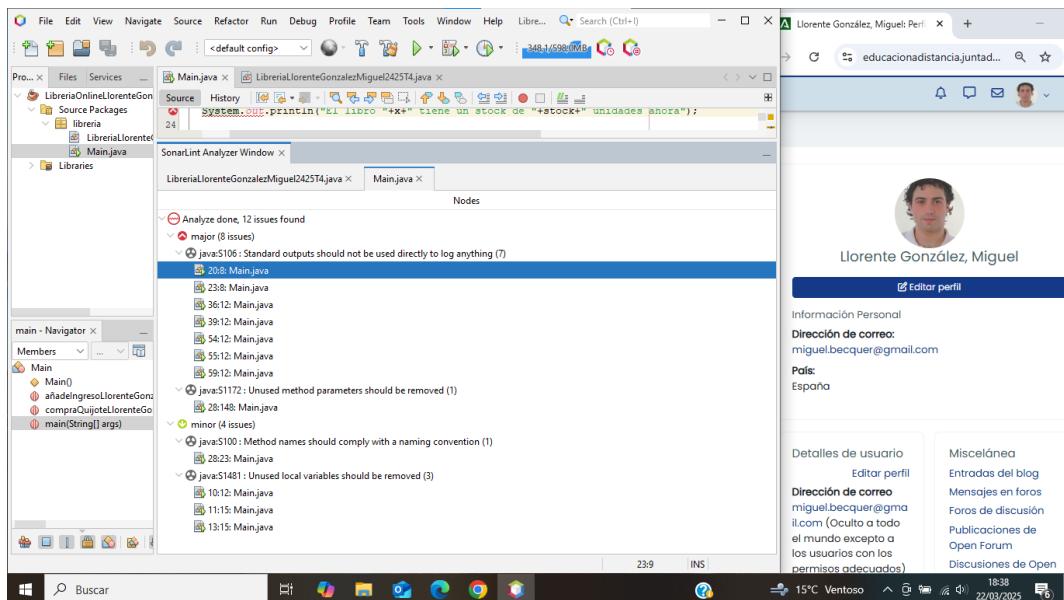


7- Analiza el código del proyecto de la tarea con el analizador de código SonarLint. Muestra y comenta los resultados.

Vamos a analizar los códigos de ambas clases. Para ello, en el panel izquierdo, hacemos clic derecho en la clase ‘LibreriaLlorenteGonzalezMiguel2425T4’ y seleccionamos ‘Tools>Analyze with SonarLint’. En este caso, el análisis nos indica que al escribir códigos que puedan lanzar excepciones, estas no deben ser genéricas (debemos escribir una excepción concreta; IllegalStateException, etc.)



Hacemos lo mismo con la clase Main. En este caso, se nos advierte del uso de ‘System.out.println()’ para logging (informar del estado del programa y de su ejecución). También nos avisa que hay un parámetro que no estamos usando, ‘concepto’, creado en el apartado 5.

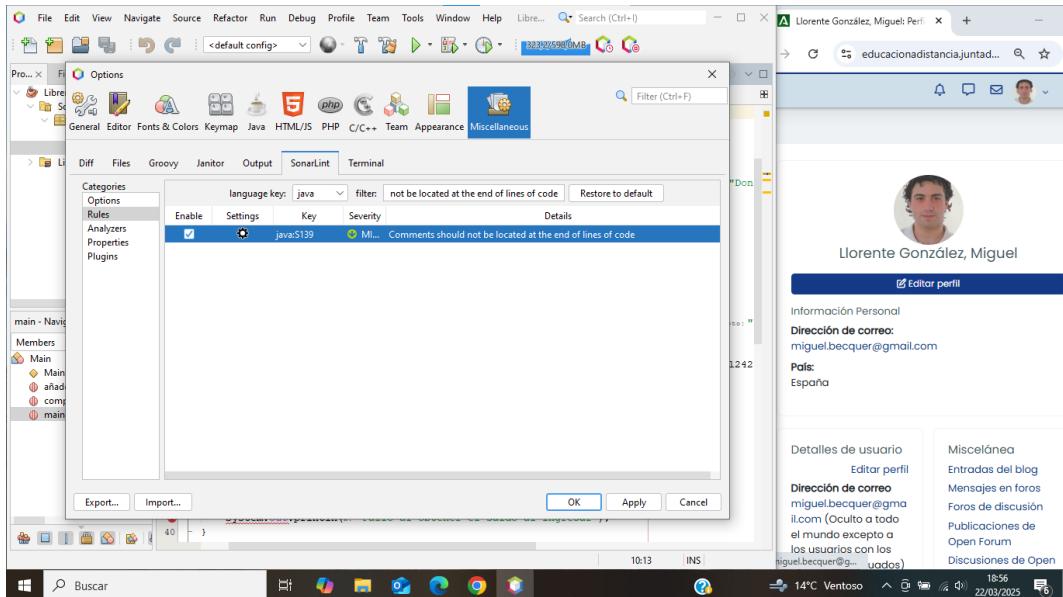


Además nos recuerda, con menor importancia, que debemos ceñirnos al convenio para la elección de nombres de los métodos. Esto se debe a que el método ‘añadeIngresoLlorenteGonzalezMiguel2425T4’ contiene una ‘ñ’. Por último, y también con menor importancia, se nos indica que hay variables declaradas que no se están

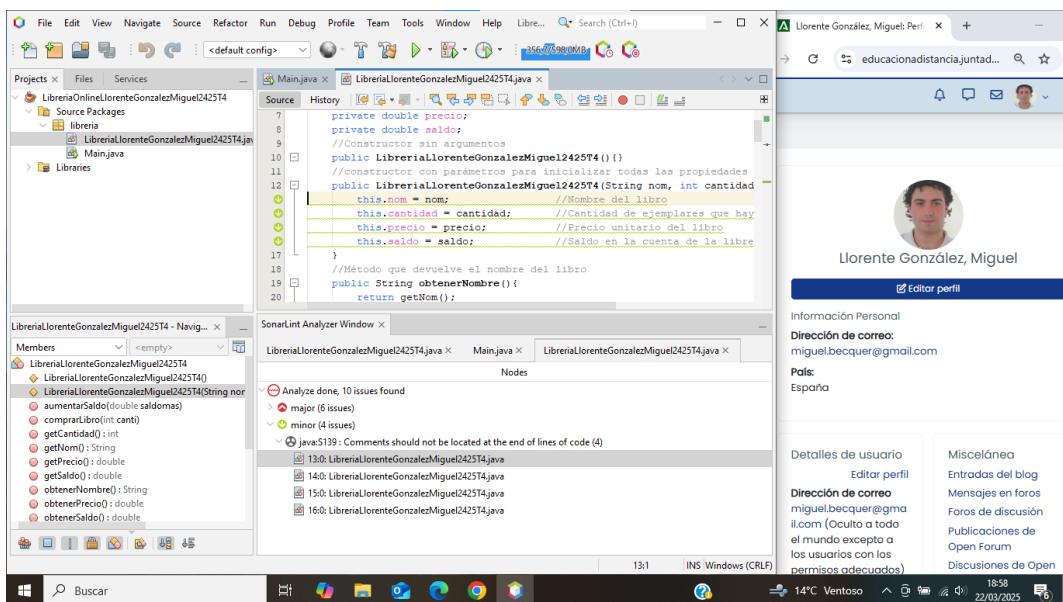
usando. Esto se debe a que se declararon antes de crear los métodos de los ejercicios 2 y 3, que ya llevan las declaraciones de sus propias variables en sus respectivos códigos, por lo que ahora no son necesarias en el código general.

8- Añade en SonarLint la regla Comments should not be located at the end of lines of code. Analiza el nuevo y, muestra y comenta los resultados.

En la pestaña ‘Tools’, escogemos esta vez la opción ‘Options’ (Opciones). En la ventana saliente, debemos situarnos en la pestaña ‘Miscellaneous’ y, en ésta, escogemos la pestaña ‘SonarLint’. Una vez aquí, podemos escoger la pestaña ‘Rules’ en el panel izquierdo, donde tenemos diversas reglas que usa el Plugin. Escribimos en el campo de búsqueda la regla indicada y la activamos:



Volvemos a analizar el primer código, la clase ‘LibreriaLlorenteGonzalezMiguel2425T4’:



A parte de las advertencias anteriores, se nos indica que en cuatro ocasiones se ha escrito código al final de líneas de código.

En la clase ‘Main’ sucede lo mismo; salen las mismas advertencias que antes, más que hay dos líneas de código con comentarios al final:

The screenshot shows an IDE interface with several windows open. On the left, the 'Projects' view shows a project named 'LibreriaOnlineLlorenteGonzalezMiguel2425T4'. The main editor window displays 'Main.java' with the following code:

```

34     ingresa = 30.5; //Ingresa que se va a realizar en el saldo de la libreria
35     milLibreriaLlorenteGonzalezMiguel12425T4.aumentarSaldo(milLibreriaLlorenteGonzalezMiguel12425T4.obtenerSaldo());
36     System.out.println("El saldo actual despues de ingresar es de "+saldoActual);
37 } catch (Exception e)
38 {
39     System.out.println("Falló al obtener el saldo al ingresar");
40 }
41
42 public static void compraQuijoteLlorenteGonzalezMiguel12425T4(LibreriaLlorenteGonzalezMiguel12425T4 libreria, String libro, double dinero, double saldoActual) {
43     String x;
44     int num;
45     double dinero;
46     double saldoActual;
47     try {
48         x = "Don Quijote de la Mancha";
49         num = 2; //Número de ejemplares a comprar
50     }
51 }
52
53 
```

The 'SonarLint Analyzer Window' shows several issues:

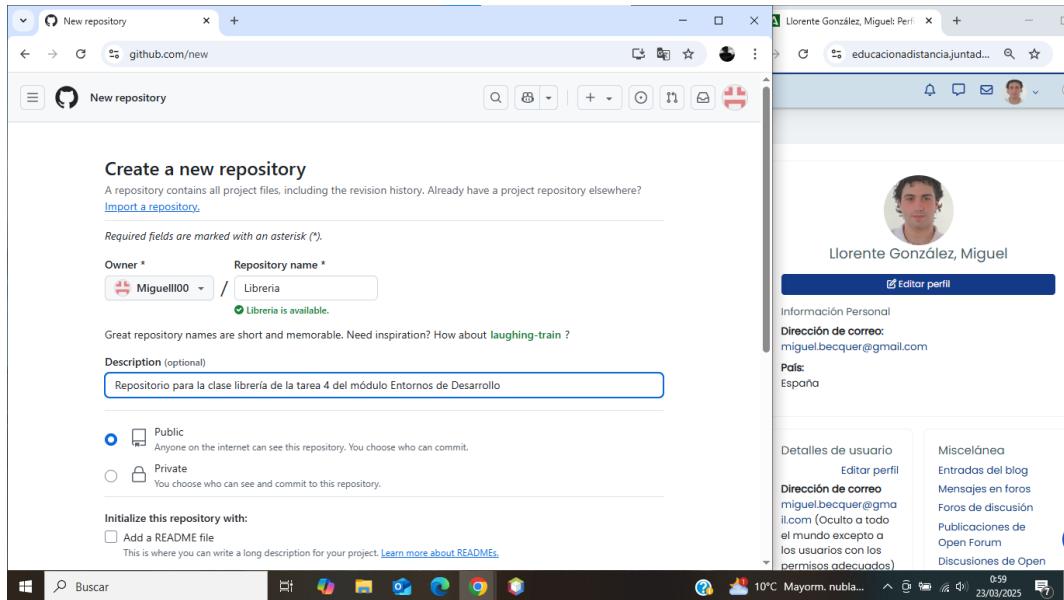
- major (8 issues)
- minor (6 issues)
- javaS100: Method names should comply with a naming convention (1)
- javaS139: Comments should not be located at the end of lines of code (2)
- javaS1481: Unused local variables should be removed (3)

On the right, a GitHub profile for 'Llorente González, Miguel' is displayed, showing basic information like email and location.

Control de versiones con Git.

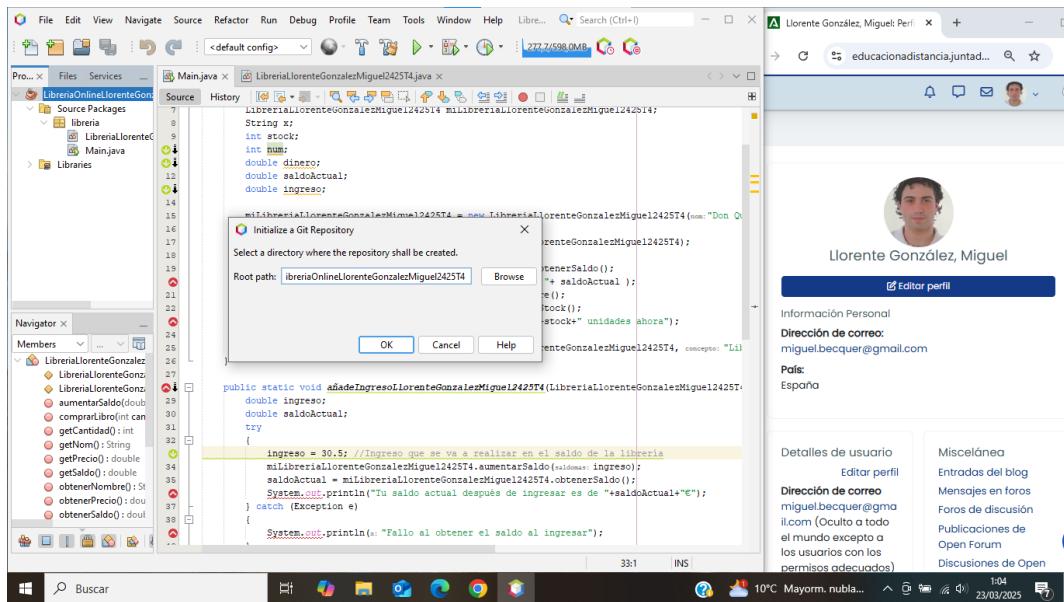
9- Inicializa el control de versiones con Git y haz un commit con el comentario “Mi primer commit XXX”.

Primero, creamos un repositorio en GitHub:

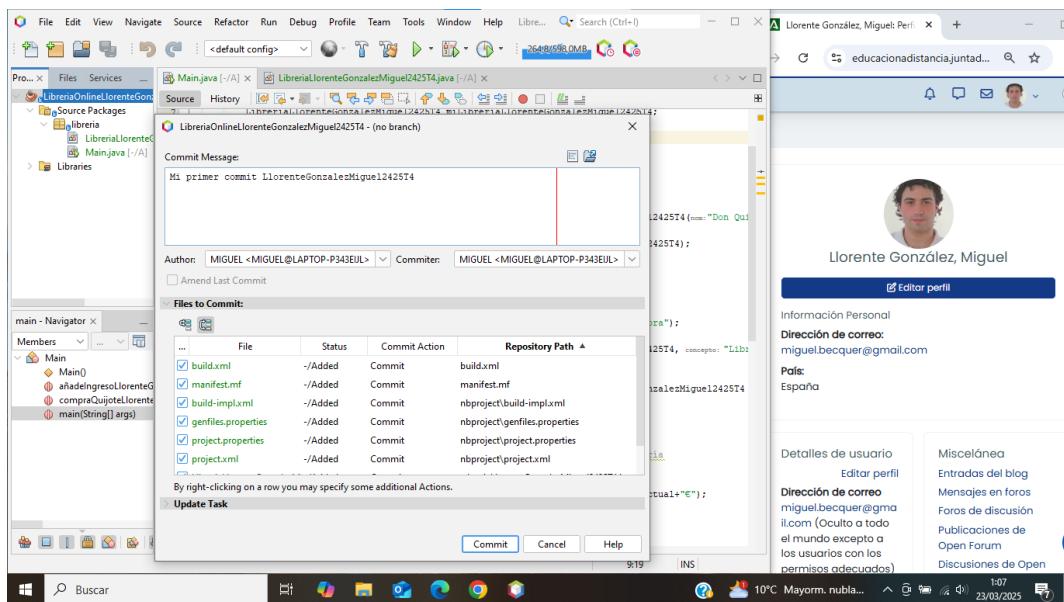


Ahora, inicializamos el repositorio desde la pestaña ‘Team’ seleccionando ‘Git>Initialize Repository...’

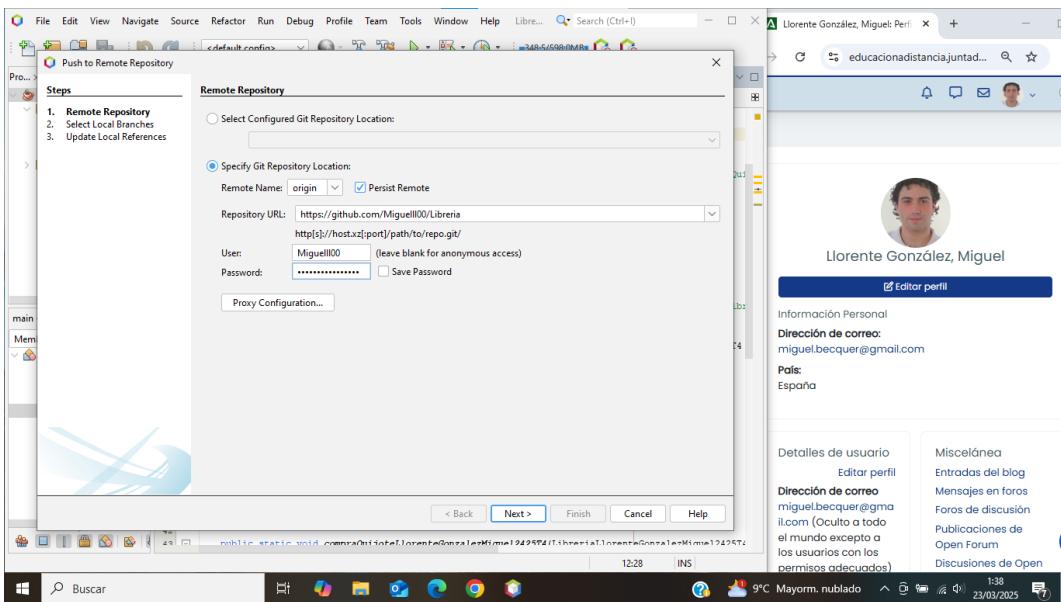
Se abrirá una ventana donde tendremos que verificar la ruta del repositorio:



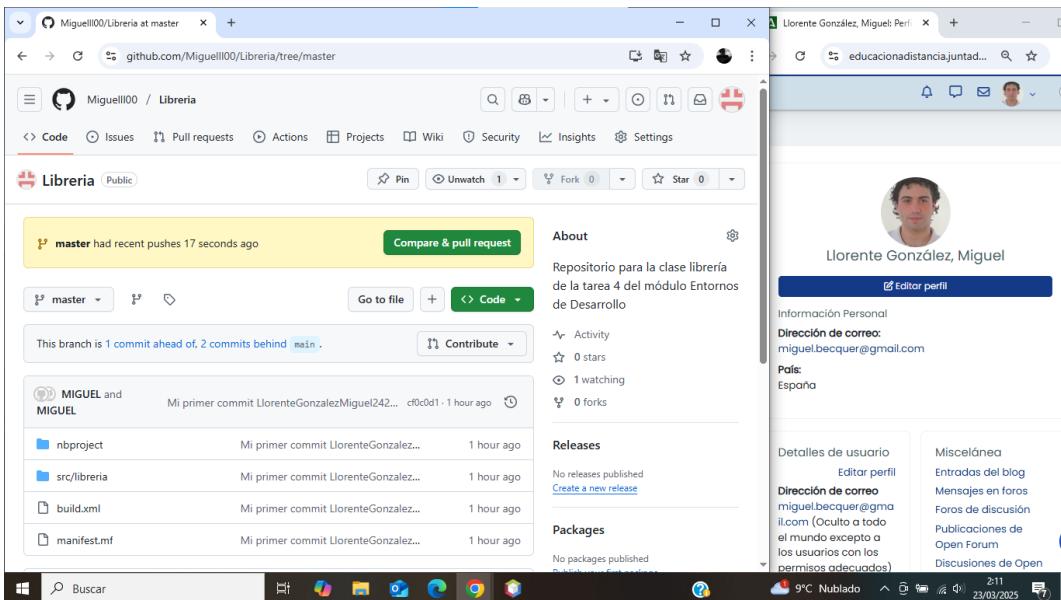
Ahora, en la misma pestaña ‘Team’ de antes, podemos ver que las opciones son distintas. Elegimos ‘Commit’. Se abre una ventana donde escribiremos el comentario indicado, y se pueden escoger qué archivos irán al repositorio (yo los dejo todos):



Ahora, también en la pestaña ‘Team’, seleccionamos la opción ‘Remote>Push’. Se abre una ventana, en la que pegamos la dirección del repositorio creado en GitHub. También escribo mi nombre de usuario y una clave que he generado en los ajustes de GitHub para poder acceder desde NetBeans. Hacemos clic en ‘Next’, elegimos la única casilla que se nos muestra y volvemos a hacer clic en ‘Next’, y luego en ‘Finish’ :



Ahora tenemos el proyecto en el repositorio:



Documenta el código con las facilidades que da el IDE NetBeans.

10- Comenta cada uno de los elementos principales de la clase Libreria y de la clase Main siguiendo las normas de javadoc.

Empiezo con la clase Libreria. Lo primero que incluyo es, al inicio, un comentario con una explicación de la clase y el nombre del autor; yo. Luego, cambio el comentario sobre el constructor con todos los parámetros de manera que doy información sobre cada uno de ellos:

The screenshot shows the NetBeans IDE interface. On the left is the Project Explorer with a package named 'LibreriaOnlineLlorenteGonzalez'. In the center is the Source Editor for 'Main.java' containing Java code with Javadoc comments. On the right is a browser window displaying a user profile for 'Llorente González, Miguel' with fields for personal information like email and location, and sections for user details and miscellanea.

```
package libreria;

/*
 * Crear y manejar librerías. Es una clase que nos permite crear instancias de libro y, gracias a sus métodos, obtener información de ellas y/o poder modificarlas.
 * Author: Miguel Llorente González
 */

public class LibreriaLlorenteGonzalezMiguel2425T4 {
    private String nom;
    private int cantidad;
    private double precio;
    private double saldo;

    //Constructor sin argumentos
    public LibreriaLlorenteGonzalezMiguel2425T4() {
        /**
         * Constructor con parámetros para inicializar todas las propiedades del libro.
         * @param nom Nombre del libro.
         * @param cantidad Cantidad de ejemplares en stock.
         * @param precio Precio unitario del libro.
         * @param saldo Saldo en la cuenta de la librería.
         */
        public LibreriaLlorenteGonzalezMiguel2425T4(String nom, int cantidad, double precio,
            this.nom = nom; //Nombre del libro
            this.cantidad = cantidad; //Cantidad de ejemplares que hay en stock
            this.precio = precio; //Precio unitario del libro
            this.saldo = saldo; //Saldo en la cuenta de la librería
        }
    }
}
```

Ahora, para los métodos ‘getters’ que estaban ya escritos (que usan los que se crearon por refactorización) escribimos una descripción y especificamos qué devolverá:

The screenshot shows the NetBeans IDE interface. The code has been updated with detailed Javadoc comments for each getter method, specifying the return type and what it represents. The browser window on the right shows the same user profile as before.

```
    /**
     * Método que devuelve el nombre del libro.
     * @return Nombre del libro.
     */
    public String obtenerNombre() {
        return getNom();
    }

    /**
     * Método que devuelve el precio del libro.
     * @return Precio del libro
     */
    public double obtenerPrecio() {
        return getPrecio();
    }

    /**
     * Método que devuelve la cantidad de libros de ese ejemplar que quedan para vender.
     * @return Cantidad en stock.
     */
    public int obtenerStock() {
        return getCantidad();
    }

    /**
     * Método para obtener el saldo en la cuenta.
     * @return Saldo en la cuenta
     */
    public double obtenerSaldo() {
        return getSaldo();
    }
}
```

Ahora hago lo mismo para los dos métodos que no son setters ni getters, que necesitan parámetros; mantengo sus descripciones y añado una explicación para sus parámetros:

```

68     /**
69      * Método para <b>aumentar</b> el saldo en un número indicado.
70      * @param saldoMas <b>diner</b> en que el sueldo va a <b>aumentar</b>.
71      * @throws Exception
72      */
73     public void aumentarSaldo(double saldoMas) throws Exception{
74         if (saldoMas<0)
75             throw new Exception("No se puede ingresar 0 o una cantidad negativa");
76         setSaldo(getSaldo() + saldoMas);
77     }
78
79     /**
80      * Método para <b>comprar</b> un ejemplar de un libro, <b>actualiza</b> el stock y el saldo.
81      * @param cantidad <b>libros</b> de libros a comprar.
82      * @throws Exception
83      */
84     public void comprarLibro(int cantidad) throws Exception{
85         if (cantidad <= 0)
86             throw new Exception("No se puede retirar una cantidad negativa de libros");
87         if (getCantidad()>cantidad){
88             if (getSaldo() < getPrecio())
89                 throw new Exception("No hay suficiente saldo para comprar");
90             else{
91                 setCantidad(getCantidad() - cantidad);
92                 setSaldo(getSaldo() + (getPrecio() * cantidad));
93             }
94         }
95         else
96             throw new Exception("No se pueden comprar más libros de los que hay disponibles");
97     }
98 }

```

Por último, escribo los comentarios de Javadoc de los métodos que se crearon mediante refactorización:

```

101    /**
102     * Método que devuelve el nombre del libro.
103     * @return Nombre del libro.
104     */
105    public String getNom() {
106        return nom;
107    }
108
109    /**
110     * Método que establece un cambio en el nombre del libro.
111     * @param nom Nuevo nombre.
112     */
113    public void setNom(String nom) {
114        this.nom = nom;
115    }
116
117    /**
118     * Método que devuelve la cantidad de libros de ese ejemplar que quedan para vender.
119     * @return Cantidad en stock.
120     */
121    public int getCantidad() {
122        return cantidad;
123    }
124
125    /**
126     * Método que establece una nueva cantidad de libros de ese ejemplar que quedan para vender.
127     * @param cantidad Nuevo valor de la cantidad.
128     */
129    public void setCantidad(int cantidad) {
130        this.cantidad = cantidad;
131    }

```

Para la clase Main hago lo mismo. Primero escribo una descripción de la clase con el nombre del autor. Luego, escribo una descripción para el método main:

The screenshot shows an IDE interface with the Main.java file open. The code includes a class-level Javadoc comment for the Main class and a method-level Javadoc comment for the main method. To the right, a browser window displays a user profile for 'Llorente González, Miguel' with details like email and location.

```

1 package libreria;
2 /**
3  * Clase que hace uso de la clase LibreriaLlorenteGonzalezMiguel12425T4. Consta de:
4  * <ol>
5  *   <li>El método <b>main</b> que usa los métodos de la clase <b>Libreria</b>.</li>
6  *   <li>Dos <b>nuevos métodos</b>.</li>
7  * </ol>
8  * @author Miguel Llorente González
9  */
10 public class Main {
11
12     /**
13      * Método <b>main</b> que realiza operaciones con la clase Libreria y con los métodos
14      * <b>creados en esta misma clase</b>.
15      * @param args
16      */
17     public static void main(String[] args) {
18         LibreriaLlorenteGonzalezMiguel12425T4 miLibreriaLlorenteGonzalezMiguel12425T4;
19         String x;
20         int stock;
21         int num;
22         double dinero;
23         double saldoActual;
24         double ingreso;
25
26         miLibreriaLlorenteGonzalezMiguel12425T4 = new LibreriaLlorenteGonzalezMiguel12425T4();
27
28         compraQuijoteLlorenteGonzalezMiguel12425T4(miLibreriaLlorenteGonzalezMiguel12425T4);
29
30         saldoActual = miLibreriaLlorenteGonzalezMiguel12425T4.obtenerSaldo();
31
32     }
33 }

```

Acabo escribiendo la descripción para los métodos creados en la clase Main mediante la refactorización de código:

The screenshot shows the IDE with the Main.java file open, showing the refactored code where the new methods have been annotated with Javadoc comments. The browser window next to it shows the same user profile as before.

```

41 /**
42  * Añade <b>30.5€</b> a un objeto de la clase LibreriaLlorenteGonzalezMiguel12425T4
43  * introduciémos como parámetro. También tendremos un parámetro <b>concepto</b>.
44  * @param miLibreriaLlorenteGonzalezMiguel12425T4
45  * @param concepto
46  */
47 public static void añadeIngresoLlorenteGonzalezMiguel12425T4(LibreriaLlorenteGonzalezMiguel12425T4
48     double ingreso;
49     double saldoActual;
50     try
51     {
52         ingreso = 30.5; //Ingreso que se va a realizar en el saldo de la librería
53         miLibreriaLlorenteGonzalezMiguel12425T4.aumentarSaldo(+ingreso);
54         saldoActual = miLibreriaLlorenteGonzalezMiguel12425T4.obtenerSaldo();
55         System.out.println("Tu saldo actual después de ingresar es de "+saldoActual);
56     }
57     catch (Exception e)
58     {
59         System.out.println("Fallo al obtener el saldo al ingresar");
60     }
61 }
62
63 /**
64  * Realiza la <b>compra</b> de <b>dos</b> libros a un objeto de la clase LibreriaLlor
65  * Se debería usar en una instancia de objeto que tenga como valor 'nom' un String <b>
66  * </b>.
67  * @param miLibreriaLlorenteGonzalezMiguel12425T4
68  */
69 public static void compraQuijoteLlorenteGonzalezMiguel12425T4(LibreriaLlorenteGonzalezMiguel12425T4
70     String x;
71     int num;
72     double dinero;
73 }

```

11- Genera documentación Javadoc para todo el proyecto.

Para generar el documento Javadoc hacemos clic derecho sobre el proyecto y escogemos la opción ‘Generate Javadoc’. Tras cargar, se abre una ventana de un navegador en el que podemos ver la información de ambas clases en sus respectivas páginas en el Javadoc creado:

Class LibreriaLlorenteGonzalezMiguel2425T4

java.lang.Object
libreria.LibretillorenteGonzalezMiguel2425T4

public class **LibreriaLlorenteGonzalezMiguel2425T4** extends Object

Crear y manejar librerías. Es una clase que nos permite crear instancias de **librerías** y, gracias a sus métodos, obtener información de ellas y/o poder modificarlas.

Constructor Summary

Constructors	Description
<code>LibreriaLlorenteGonzalezMiguel2425T4()</code>	Constructor con parámetros para inicializar todas las propiedades de la clase.

Method Summary

All Methods	Instance Methods	Concrete Methods
Modifier and Type	Method	Description
void	<code>aumentarSaldo(double saldoMas)</code>	Método para aumentar el saldo en un número indicado.
void	<code>comprarLibro(int cantidad)</code>	Método para comprar un ejemplar de un libro, actualiza el stock y el saldo.
int	<code>getCantidad()</code>	Método que devuelve la cantidad de libros de ese ejemplar que quedan para vender.
String	<code>getNombre()</code>	Método que devuelve el nombre del libro.
double	<code>getPrecio()</code>	Método que devuelve el precio del libro.

Class Main

java.lang.Object
libreria.Main

public class **Main** extends Object

Clase que hace uso de la clase **LibreriaLlorenteGonzalezMiguel2425T4**. Contiene:

- 1. Un método `main` que usa los métodos de la clase **Libreria**.
- 2. Dos nuevos métodos.

Constructor Summary

Constructors	Description
<code>Main()</code>	

Method Summary

All Methods	Static Methods	Concrete Methods
Modifier and Type	Method	Description
static void	<code>añadeInventarioLlorenteGonzalezMiguel2425T4(LibreriaLlorenteGonzalezMiguel2425T4 libreriaLlorenteGonzalezMiguel2425T4, String[] conceptos)</code>	Añade 30,5€ a un objeto de la clase LibreriaLlorenteGonzalezMiguel2425T4 que introduciremos como parámetro.
static void	<code>compraLibro(LibreriaLlorenteGonzalezMiguel2425T4 libreriaLlorenteGonzalezMiguel2425T4, int cantidad)</code>	Realiza la compra de dos libros a un objeto de la clase LibreriaLlorenteGonzalezMiguel2425T4 .
static void	<code>main(String[] args)</code>	Método <code>main</code> que realiza operaciones con la clase Libreria y con

Control de versiones con Git

12- Hacer Commit con el comentario “Mi proyecto XXX comentado con javadoc”.

Una vez hemos creado el proyecto con Javadoc, hacemos commit al igual que en el apartado ‘9’. Ahora, los códigos en el repositorio GitHub están comentados con Javadoc:

