

Tarea Unidad 4

Optimización y documentación

Entornos de Desarrollo

Curso DAM 2024/25

Calificación obtenida: 10/10

Índice

Detalles de la tarea	3
Ejercicio 1: Descargar el proyecto inicial	4
Ejercicio 2: Refactorización del código	6
2.1. Mover las clases al paquete cuentas	6
2.2. Renombrar la variable miCuenta	7
2.3. Crear el método operativa_cuenta	10
2.4. Encapsular los atributos de la clase CCuenta	12
2.5. Modificar el método operativa_cuenta	15
Ejercicio 3: Configuración de Git y GitHub	17
3.1. Inicializar Git y crear un repositorio	17
3.2. Realizar un commit descriptivo	24
3.3. Mostrar el historial de versiones	27
Ejercicio 4: Documentación con Javadoc	28
4.1. Insertar comentarios Javadoc	28
4.2. Generar documentación Javadoc	30
Conclusión	34

Detalles de la tarea

En esta tarea, trabajarás con el proyecto Java "**Deposito**", que incluye las clases CCuenta y Main. Tu objetivo será realizar tareas de refactorización, documentación y configuración de Git/GitHub, siguiendo instrucciones claras y ordenadas. **Deberás numerar cada ejercicio según los pasos indicados, encabezando cada uno con su número y enunciado**, y entregar el documento en formato PDF para facilitar la corrección.

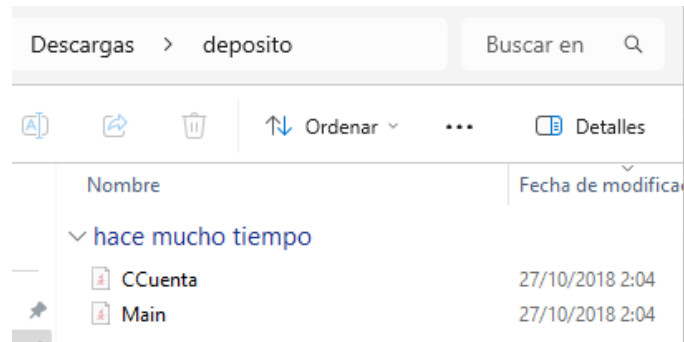
Cada ejercicio debe incluir:

1. El número y enunciado del ejercicio como encabezado.
2. Explicaciones claras de lo que has hecho, por qué lo hiciste y qué lograste.
3. Capturas de pantalla legibles que demuestren cada paso.
4. Código relevante si aplica (con formato adecuado en el documento).

Ejercicio 1: Descargar el proyecto inicial

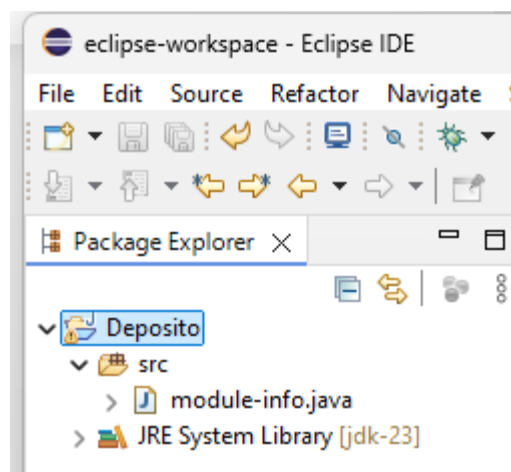
Descarga el archivo Deposito.zip, descomprímelo y verifica que contiene las clases CCuenta y Main

Abre el archivo proporcionado, extrae su contenido y asegúrate de que incluye los archivos mencionados



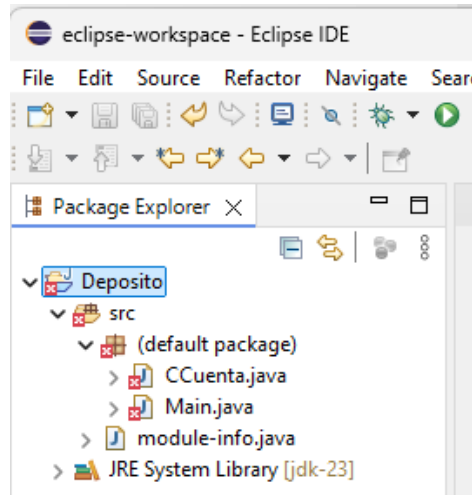
Crear un nuevo proyecto Java en Eclipse

1. Abrir Eclipse
2. Ir al menú superior y seleccionar: **File** → **New** → **Java Project**.
3. En la ventana que aparece:
 - **Project Name:** Escribir **Deposito**.
 - Asegurarse de que la opción "Use default location" está marcada.
 - Hacer clic en **Finish**.
4. Eclipse crea un proyecto con la estructura básica.



Agregar los archivos Java

1. Localizar los archivos Java guardados en mi ordenador.
2. Arrastrar y soltar los archivos Java directamente en la carpeta src dentro del proyecto Deposito.



Explicaciones claras de lo que has hecho, por qué lo hiciste y qué lograste

¿Qué hemos hecho?	¿Por qué lo hemos hecho?	¿Qué hemos logrado?
Abrir Eclipse	Porque es un entorno que facilita la organización y el desarrollo de proyectos Java	Iniciamos la herramienta donde vamos a crear y gestionar el proyecto Deposito
Crear el proyecto	Un proyecto Java es un contenedor que organiza y gestiona archivos, configuraciones y librerías. Sin un proyecto, Eclipse no podrá entender ni compilar los archivos Java.	Creamos el proyecto llamado Deposito , el cual tiene una estructura básica: <ul style="list-style-type: none"> • Una carpeta <code>src</code> (para el código fuente). • Configuración predeterminada para la versión de Java que usa el proyecto.
Localizar y agregar los archivos Java	Como ya tenemos archivos Java creados fuera de Eclipse, necesitamos agregarlos al proyecto para que Eclipse los reconozca y trabaje con ellos. Al mover los archivos al proyecto (dentro de la carpeta <code>src</code>), Eclipse los reconoce y los trata como parte del código fuente del proyecto.	Identificamos los archivos Java que queremos incluir en el proyecto. Los archivos Java ahora están dentro de la carpeta (<code>src</code>).

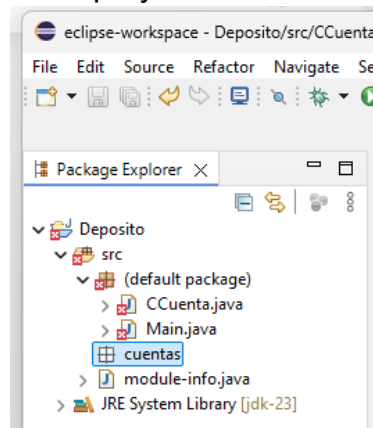
Ejercicio 2: Refactorización del código

2.1. Mover las clases al paquete cuentas

Organiza las clases en un paquete llamado cuentas

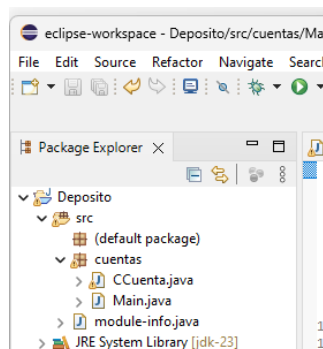
Crea un paquete en tu proyecto llamado cuentas

1. Clic derecho sobre la carpeta `src` del proyecto dentro del Explorador de proyectos en Eclipse.
2. Seleccionar: **New** → **Package**.
3. En el campo **Name**, escribir `cuentas`.
4. Clic en **Finish**.
5. Se ha creado el paquete dentro del proyecto.



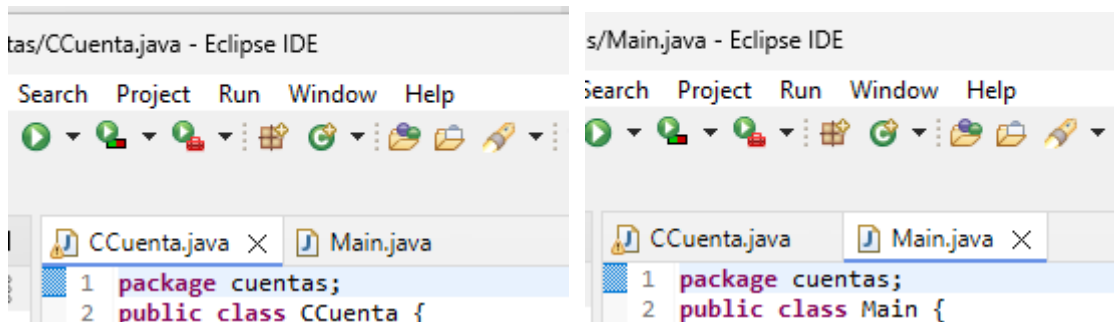
Mueve las clases CCuenta y Main a este paquete

1. Localizar las clases `CCuenta` y `Main` en el proyecto, dentro de la carpeta `src`.
2. Arrastrar cada clase desde su ubicación actual hacia el paquete `cuentas` en el Explorador de proyectos.



Asegúrate de que las referencias en el código están actualizadas

1. Abrir las clases `CCuenta` y `Main` desde el nuevo paquete `cuentas`.
2. Comprobar que la primera línea de cada archivo tenga la declaración del paquete correcta.



Explicaciones claras de lo que has hecho, por qué lo hiciste y qué lograste

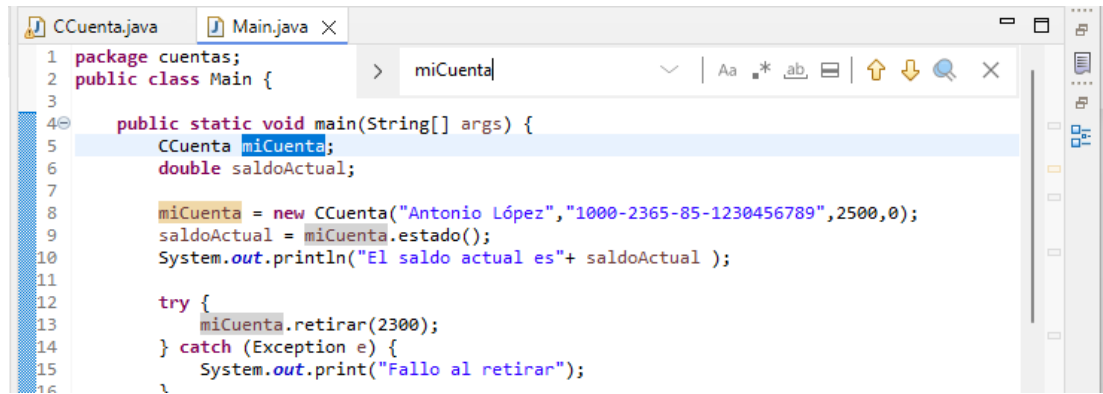
¿Qué hemos hecho?	¿Por qué lo hemos hecho?	¿Qué hemos logrado?
Hemos creado un paquete llamado <code>cuentas</code> dentro del proyecto.	Para organizar las clases <code>CCuenta</code> y <code>Main</code> en un contenedor lógico y evitar el desorden.	Logramos tener un proyecto más organizado, con un paquete específico para las clases relacionadas con las cuentas.
Movimos las clases <code>CCuenta</code> y <code>Main</code> al paquete <code>cuentas</code> .	Para agrupar las clases relacionadas y mejorar la estructura del proyecto.	Las clases <code>CCuenta</code> y <code>Main</code> ahora están dentro del paquete <code>cuentas</code> , lo que facilita su gestión y mantenimiento.
Verificamos que las referencias en el código se actualicen.	Para asegurarnos de que las clases importadas y sus referencias estén actualizadas y no haya errores.	Aseguramos que las clases sean referenciadas correctamente en todo el proyecto, evitando posibles errores de compilación.

2.2. Renombrar la variable `miCuenta`

Cambia el nombre de la variable `miCuenta` a `cuenta1` en la clase `Main`.

Localiza la variable `miCuenta` en el archivo `Main.java`.

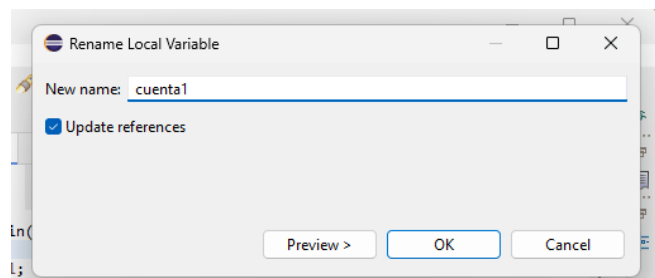
1. Abrir el archivo **Main.java**.
2. Dentro de este archivo, buscar la declaración de la variable `miCuenta` presionando `Ctrl+F` y escribiendo **miCuenta** en el campo de búsqueda para localizar todas las apariciones de la variable.
3. Asegurarme de haber encontrado la declaración de la variable (donde se crea) y también todas las referencias en el código donde se utiliza `miCuenta`.



```
1 package cuentas;
2 public class Main {
3
4     public static void main(String[] args) {
5         CCuenta miCuenta;
6         double saldoActual;
7
8         miCuenta = new CCuenta("Antonio López", "1000-2365-85-1230456789", 2500, 0);
9         saldoActual = miCuenta.estado();
10        System.out.println("El saldo actual es"+ saldoActual );
11
12        try {
13            miCuenta.retirar(2300);
14        } catch (Exception e) {
15            System.out.print("Fallo al retirar");
16        }
17    }
18 }
```

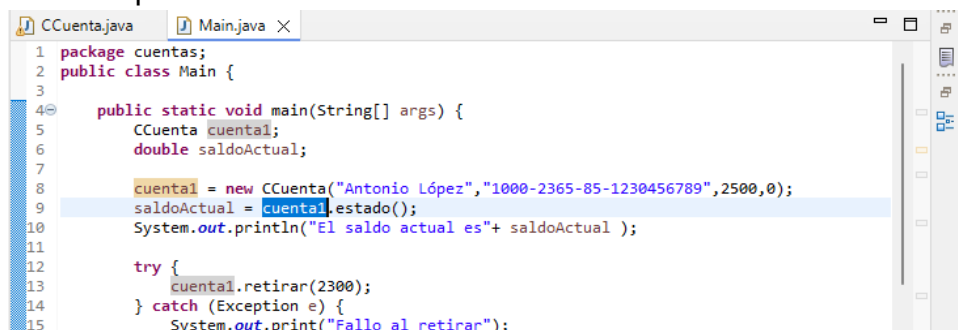
Cambia su nombre a cuenta1

1. Hacer clic sobre el nombre de la variable `miCuenta` en cualquier parte del código donde esté declarada o utilizada.
2. Con la variable `miCuenta` seleccionada, hacer clic derecho sobre ella.
3. En el menú contextual, selecciona **Refactor** → **Rename** (o usar el atajo `Shift+F6`).
4. Se habilita un cuadro de texto que permite escribir un nuevo nombre para la variable.
5. En el cuadro de texto que aparece, escribir **cuenta1** y hacer clic en **OK**.



Comprobar que la variable miCuenta ha sido renombrada

Todas las instancias de `miCuenta` han sido cambiadas a `cuenta1`, tanto en la declaración como en todas las referencias posteriores de esa variable.

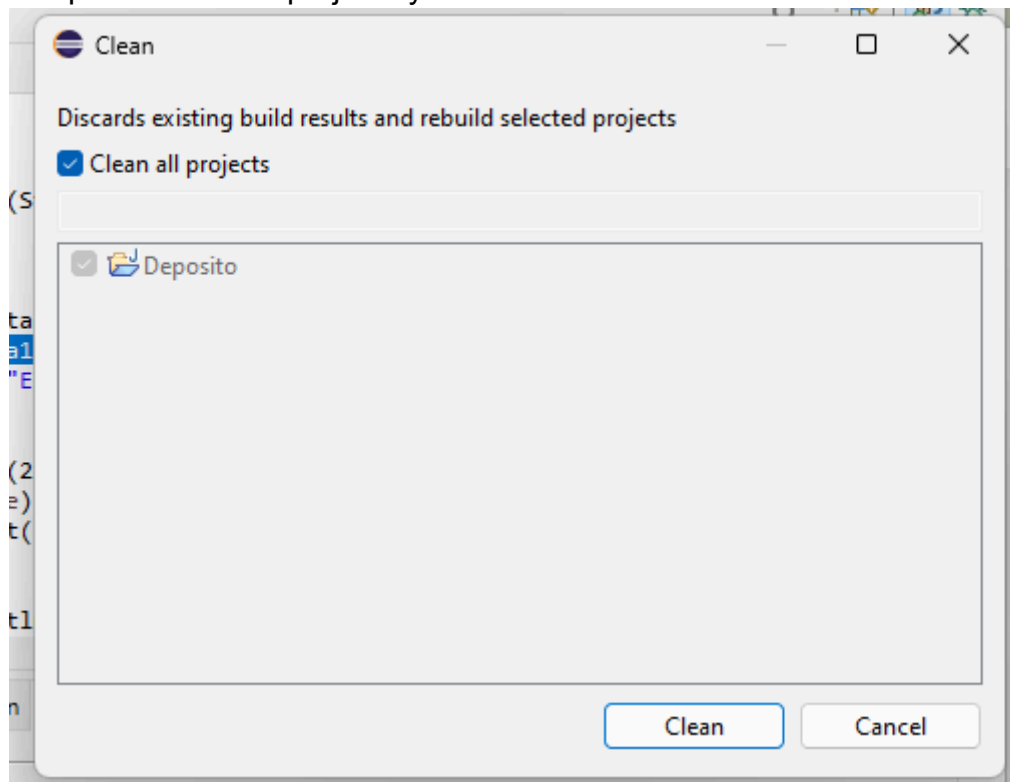


```
1 package cuentas;
2 public class Main {
3
4     public static void main(String[] args) {
5         CCuenta cuenta1;
6         double saldoActual;
7
8         cuenta1 = new CCuenta("Antonio López", "1000-2365-85-1230456789", 2500, 0);
9         saldoActual = cuenta1.estado();
10        System.out.println("El saldo actual es"+ saldoActual );
11
12        try {
13            cuenta1.retirar(2300);
14        } catch (Exception e) {
15            System.out.print("Fallo al retirar");
16        }
17    }
18 }
```

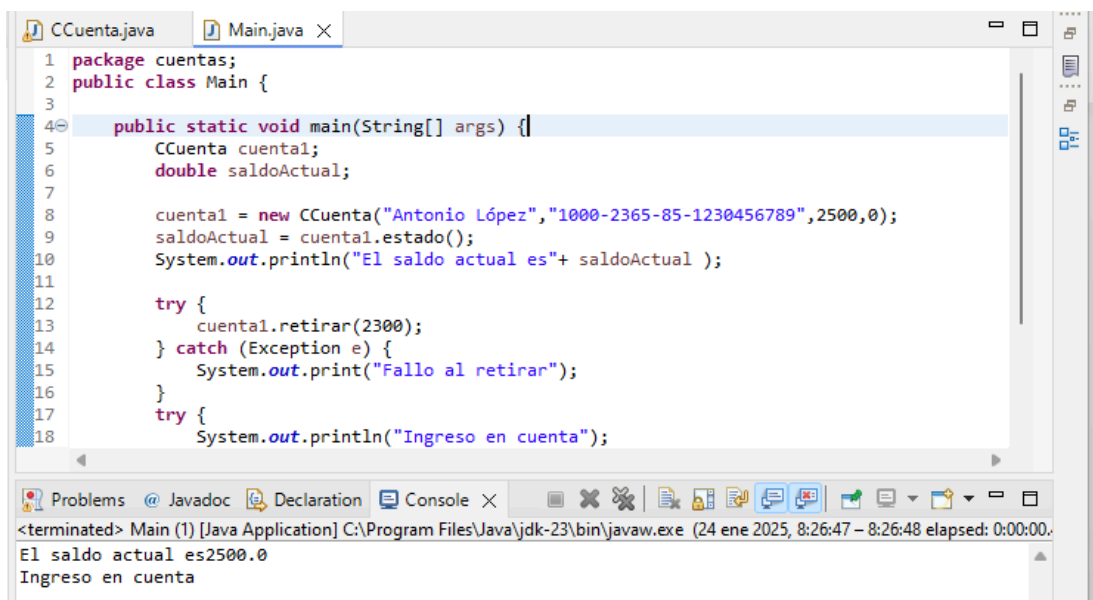
Verifica que no hay errores en la compilación

Limpiar y recompilar el proyecto

1. Limpiar y recompilar el proyecto: Ir al menú Project y seleccionar Clean.
2. Marcar la opción Clean all projects y hacer clic en OK.



El proyecto compila y se ejecuta sin problemas



Explicaciones claras de lo que has hecho, por qué lo hiciste y qué lograste

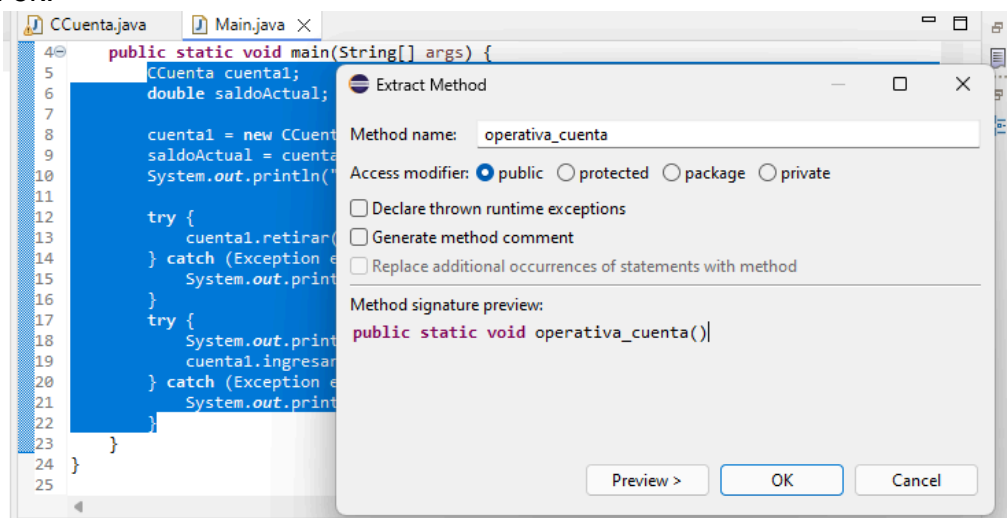
¿Qué hemos hecho?	¿Por qué lo hemos hecho?	¿Qué hemos logrado?
Hemos utilizado la funcionalidad de refactorización de Eclipse para renombrar la variable <code>miCuenta</code> a <code>cuenta1</code> .	Lo hicimos porque la refactorización permite cambiar el nombre de la variable de forma segura y eficiente, actualizando automáticamente todas las referencias.	Hemos renombrado la variable <code>miCuenta</code> a <code>cuenta1</code> en todo el código sin errores, y todas las referencias han sido actualizadas correctamente.
Seleccionamos la variable <code>miCuenta</code> y utilizamos la opción Refactor → Rename en Eclipse.	Esto se hizo para aprovechar la herramienta de refactorización, que asegura que no se omita ninguna referencia de la variable al renombrarla.	Hemos cambiado el nombre de la variable en todo el archivo sin necesidad de hacerlo manualmente en cada referencia, minimizando errores.
Limpiamos y recompilamos el proyecto para asegurarnos de que no haya errores.	Lo hicimos para verificar que el cambio se haya realizado correctamente y que el proyecto compile sin errores.	Hemos asegurado que el proyecto compile correctamente después del cambio de nombre y que no haya problemas de compilación.

2.3. Crear el método operativa_cuenta

Agrupar las operaciones realizadas con la variable `cuenta1` en un nuevo método llamado `operativa_cuenta`

Crea un método llamado `operativa_cuenta` en la clase `Main`

1. Seleccionar todo el bloque que realiza las operaciones con `cuenta1`
2. Clic derecho sobre la selección. Ir a **Refactor** → **Extract Method**.
3. Aparece un cuadro de diálogo. Escribir el nombre del nuevo método (`operativa_cuenta`) y dar a ok.



Mueve las operaciones relacionadas con `cuenta1` a este método

Se ha creado un método independiente llamado `operativa_cuenta` que encapsula todas las operaciones relacionadas con `cuenta1`.

```

1 package cuentas;
2 public class Main {
3
4     public static void main(String[] args) {
5         operativa_cuenta();
6     }
7
8     public static void operativa_cuenta() {
9         CCuenta cuenta1;
10        double saldoActual;
11
12        cuenta1 = new CCuenta("Antonio López", "1000-2365-85-1230456789", 2500, 0);
13        saldoActual = cuenta1.estado();
14        System.out.println("El saldo actual es"+ saldoActual );
15
16        try {
17            cuenta1.retirar(2300);
18        } catch (Exception e) {
19            System.out.print("Fallo al retirar");
20        }
21        try {
22            System.out.println("Ingreso en cuenta");
23            cuenta1.ingresar(695);
24        } catch (Exception e) {
25            System.out.print("Fallo al ingresar");
26        }
27    }
28 }
29

```

Asegúrate de que el método sea llamado desde main.

La parte del código que indica que el método `operativa_cuenta` es llamado desde `main` es la siguiente línea dentro del método `main`:

```

1 package cuentas;
2 public class Main {
3
4     public static void main(String[] args) {
5         operativa_cuenta();
6     }
7
8
9

```

Explicaciones claras de lo que has hecho, por qué lo hiciste y qué lograste

¿Qué hemos hecho?	¿Por qué lo hemos hecho?	¿Qué hemos logrado?
Usamos la opción de refactorización de Eclipse para extraer un método llamado <code>operativa_cuenta</code> en la clase <code>Main</code> .	Para agrupar automáticamente las operaciones relacionadas con la variable <code>cuenta1</code> en un método independiente, simplificando el proceso manual.	Creamos el método <code>operativa_cuenta</code> de manera más rápida y precisa, asegurándonos de no cometer errores manuales.
Eclipse movió todas las operaciones relacionadas	Para garantizar que las operaciones con <code>cuenta1</code> estén	Centralizamos las operaciones de la variable <code>cuenta1</code> en un

con <code>cuenta1</code> al nuevo método <code>operativa_cuenta</code> .	encapsuladas, manteniendo el código organizado y modular.	método específico, dejando el método <code>main</code> más limpio.
Confirmamos que el método <code>operativa_cuenta</code> es llamado desde <code>main</code> , verificando que Eclipse ajustó las referencias automáticamente.	Para asegurarnos de que la refactorización no rompió el flujo del programa y que el código sigue siendo funcional.	Validamos que el programa ejecuta correctamente las operaciones con <code>cuenta1</code> desde el método <code>main</code> .

2.4. Encapsular los atributos de la clase CCuenta

Encapsula los atributos de la clase `CCuenta` y crea los métodos `getter` y `setter` correspondientes.

Modifica los atributos para que sean privados

Los modificadores de acceso de los atributos ya eran privados.

```

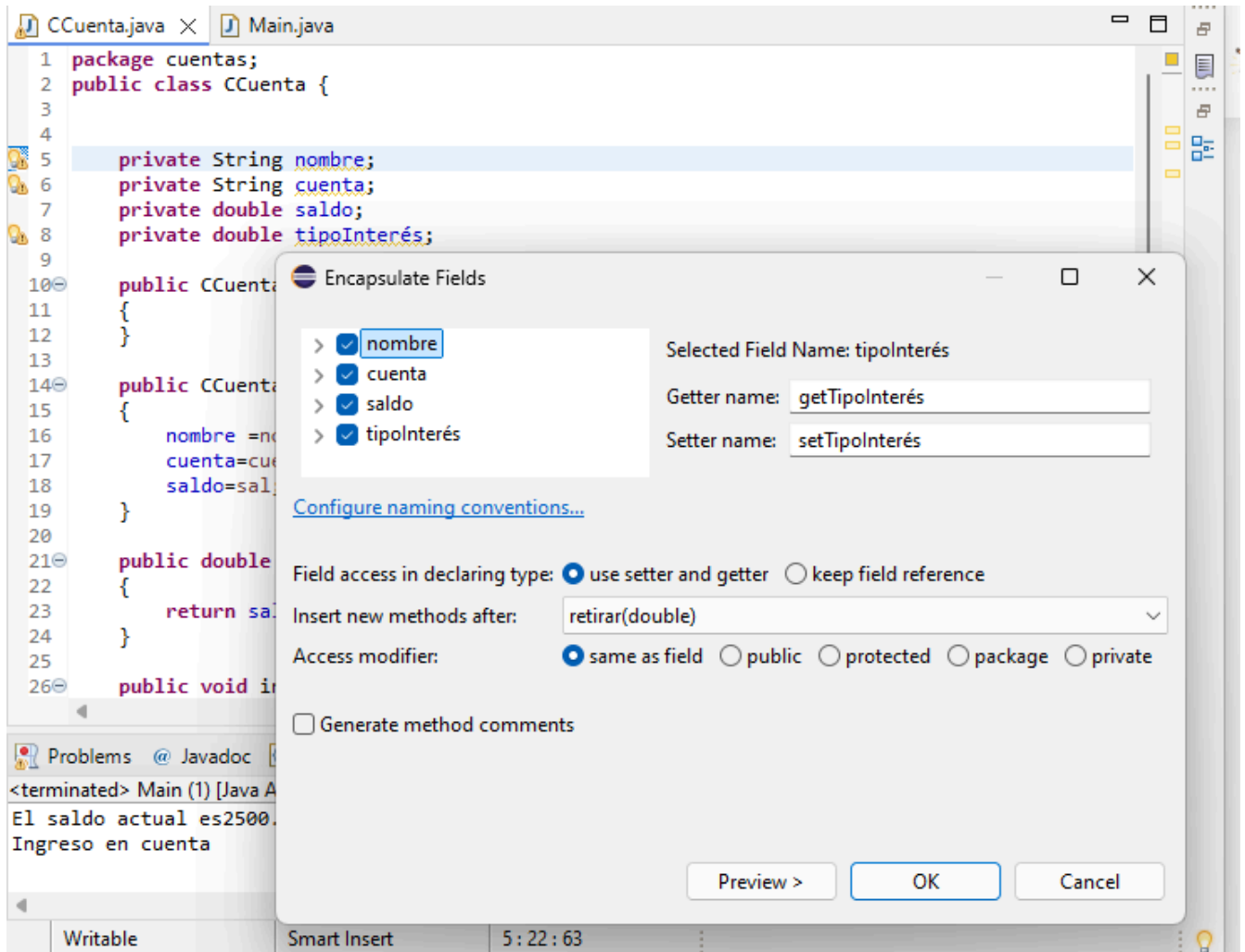
1 package cuentas;
2 public class CCuenta {
3
4
5     private String nombre;
6     private String cuenta;
7     private double saldo;
8     private double tipoInterés;
9 }

```

Crea los métodos `getter` y `setter` para cada atributo

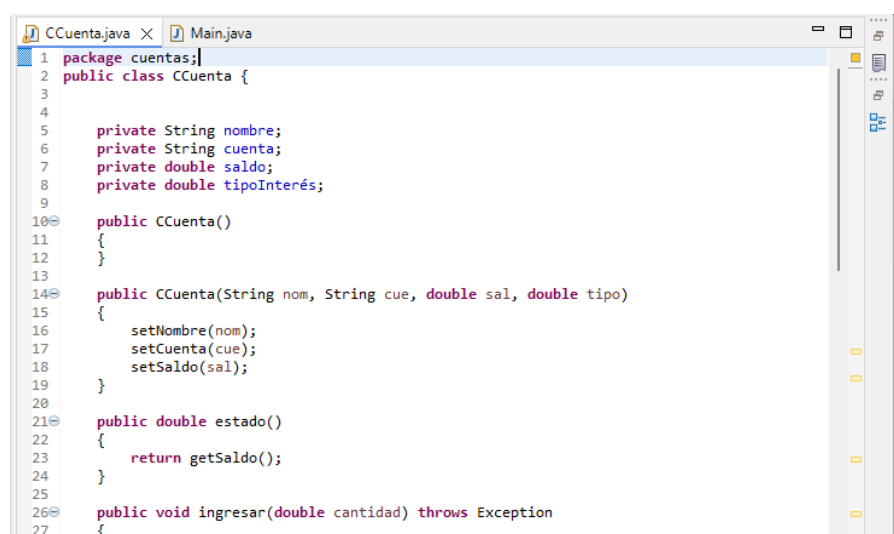
Usar refactorización para crear los métodos `getter` y `setter` para cada uno de los atributos privados

1. **Posicionar el cursor sobre un atributo** (por ejemplo, `nombre`).
2. Clic derecho y seleccionar **Refactor > Encapsulate Fields...**
3. En la ventana que aparece, **marcar todos los campos** que queremos encapsular.
4. Hacer clic en **OK**.



Código de los métodos getter y setter

Eclipse genera automáticamente los métodos getter y setter para cada atributo privado.



```

28     if (cantidad<0)
29         throw new Exception("No se puede ingresar una cantidad negativa");
30     setSaldo(getSaldo() + cantidad);
31 }
32
33 public void retirar(double cantidad) throws Exception
34 {
35     if (cantidad <= 0)
36         throw new Exception ("No se puede retirar una cantidad negativa");
37     if (estado()< cantidad)
38         throw new Exception ("No se hay suficiente saldo");
39     setSaldo(getSaldo() - cantidad);
40 }
41
42 private double getTipoInterés() {
43     return tipoInterés;
44 }
45
46 private void setTipoInterés(double tipoInterés) {
47     this.tipoInterés = tipoInterés;
48 }
49
50 private double getSaldo() {
51     return saldo;
52 }
53
54 private void setSaldo(double saldo) {
55     this.saldo = saldo;
56 }
57
58 private String getCuenta() {
59     return cuenta;
60 }
61
62 private void setCuenta(String cuenta) {
63     this.cuenta = cuenta;
64 }
65
66 private String getNombre() {
67     return nombre;
68 }
69
70 private void setNombre(String nombre) {
71     this.nombre = nombre;
72 }
73 }
74

```

Explicaciones claras de lo que has hecho, por qué lo hiciste y qué lograste

¿Qué hemos hecho?	¿Por qué lo hemos hecho?	¿Qué hemos logrado?
Comprobar que los atributos de la clase CCuenta eran privados (nombre, cuenta, saldo, tipoInterés)	Lo hemos hecho para aplicar el principio de encapsulación , protegiendo los datos internos de la clase para que no puedan ser modificados directamente desde fuera.	Los atributos de la clase CCuenta están protegidos y solo pueden ser modificados o accedidos a través de métodos.

Hemos generado los métodos getter y setter automáticamente usando refactorización en Eclipse.	Para proporcionar una forma controlada de acceder y modificar los atributos privados sin romper el principio de encapsulación.	Los métodos getter y setter permiten acceder y modificar los valores de los atributos privados de forma segura y controlada.
Usamos Refactor > Encapsulate Fields... en Eclipse para generar los métodos getter y setter para cada atributo de la clase.	Usamos refactorización para agilizar el proceso y evitar tener que escribir manualmente cada uno de los métodos getter y setter.	Se generaron los métodos automáticamente, mejorando la eficiencia y asegurando que el código sea limpio y funcional.

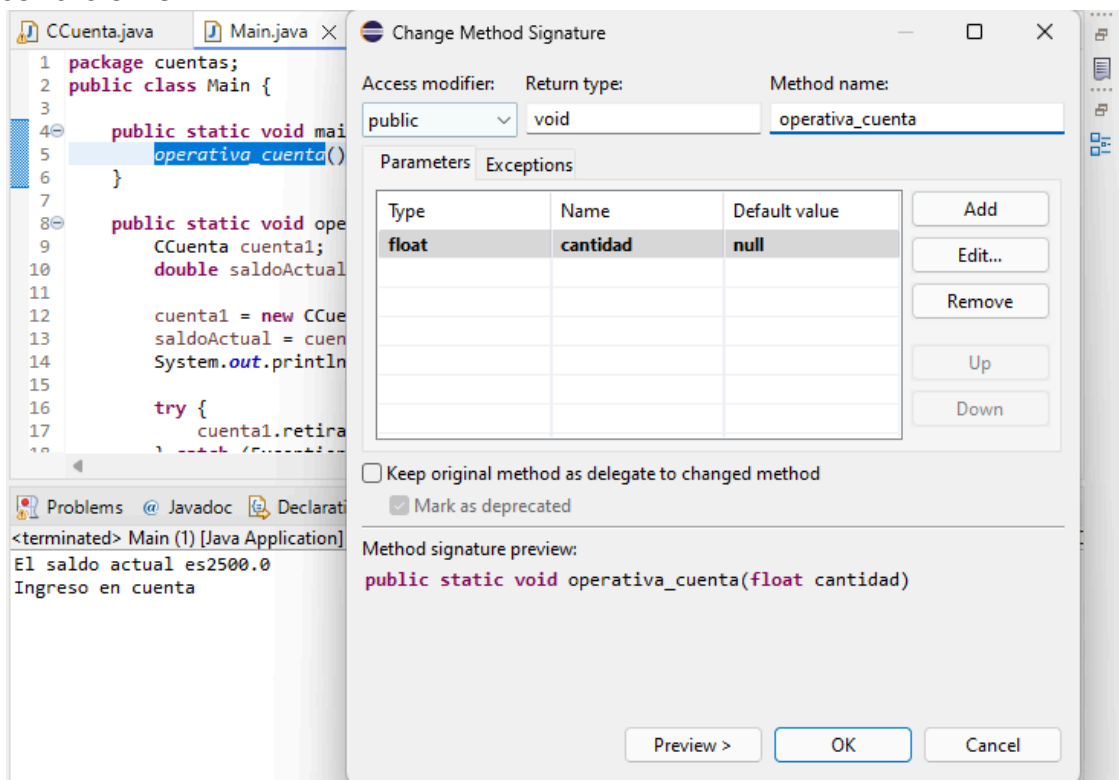
2.5. Modificar el método operativa_cuenta

Código actualizado del método mostrando el nuevo parámetro

Añade un nuevo parámetro de tipo float llamado cantidad al método operativa_cuenta

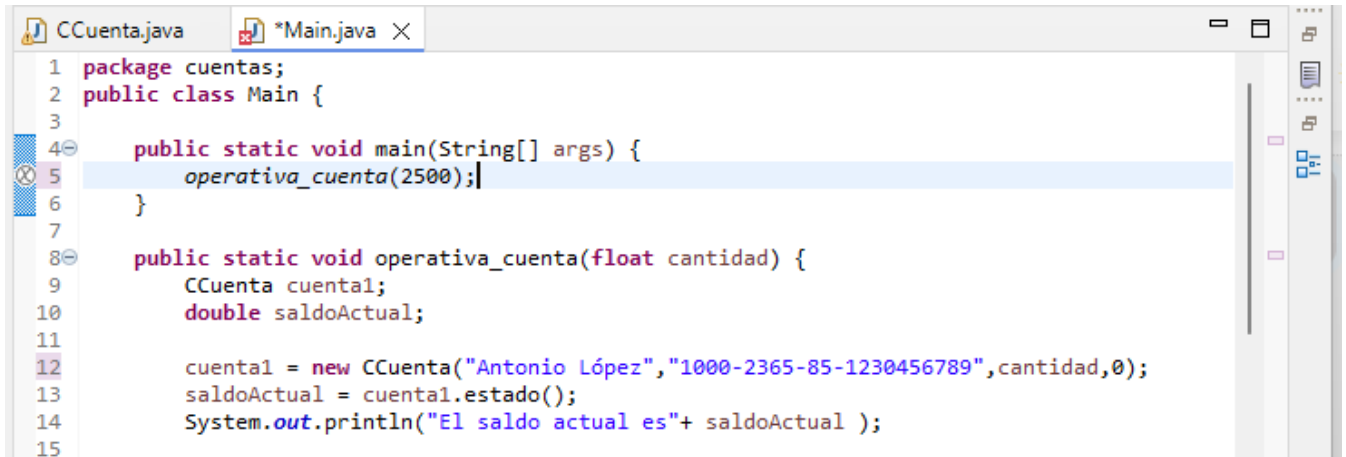
Actualiza el método operativa_cuenta para incluir el parámetro cantidad

1. Seleccionar y clic derecho sobre el nombre del método `operativa_cuenta` del archivo `Main`.
2. En el menú contextual, seleccionar: **Refactor > Change Method Signature**
3. En el cuadro de diálogo que aparece, hacer clic en **Add**
4. Especificar el tipo del nuevo parámetro como `float` y el nombre como `cantidad`.
5. Hacer clic en OK



Ajusta las llamadas al método para incluir este nuevo parámetro

Eclipse ha actualizado automáticamente todas las llamadas al método para que incluyan el nuevo parámetro



```

1 package cuentas;
2 public class Main {
3
4     public static void main(String[] args) {
5         operativa_cuenta(2500);
6     }
7
8     public static void operativa_cuenta(float cantidad) {
9         CCuenta cuenta1;
10        double saldoActual;
11
12        cuenta1 = new CCuenta("Antonio López","1000-2365-85-1230456789",cantidad,0);
13        saldoActual = cuenta1.estado();
14        System.out.println("El saldo actual es"+ saldoActual );
15    }
  
```

Explicaciones claras de lo que has hecho, por qué lo hiciste y qué lograste

¿Qué hemos hecho?	¿Por qué lo hemos hecho?	¿Qué hemos logrado?
Hemos añadido un nuevo parámetro de tipo <code>float</code> llamado <code>cantidad</code> al método <code>operativa_cuenta</code> usando la opción de refactorización en Eclipse.	Para cumplir con el requisito de agregar un parámetro que permita inicializar el saldo de la cuenta de forma dinámica, aumentando la flexibilidad del método.	Ahora el método <code>operativa_cuenta</code> puede recibir un valor externo como parámetro y usarlo para configurar el saldo inicial de la cuenta al momento de su creación.
Hemos ajustado las llamadas al método <code>operativa_cuenta</code> en el método <code>main</code> para incluir un valor representativo como argumento del nuevo parámetro.	Lo hicimos para que el programa funcione correctamente tras los cambios, asegurándonos de que las llamadas coincidan con la nueva definición del método.	Todas las llamadas al método ahora incluyen el parámetro requerido (2500 en este caso), lo que garantiza la correcta ejecución del programa.
Usamos la herramienta de Refactor > Change Method Signature... en Eclipse para modificar la firma del método y agregar el nuevo parámetro de manera rápida y eficiente.	Usamos esta herramienta para asegurarnos de que todas las referencias al método se actualizaran automáticamente, evitando errores manuales y manteniendo el código limpio.	La refactorización automática actualizó tanto la firma del método como sus referencias en el programa, ahorrando tiempo y reduciendo el riesgo de errores.

Ejercicio 3: Configuración de Git y GitHub

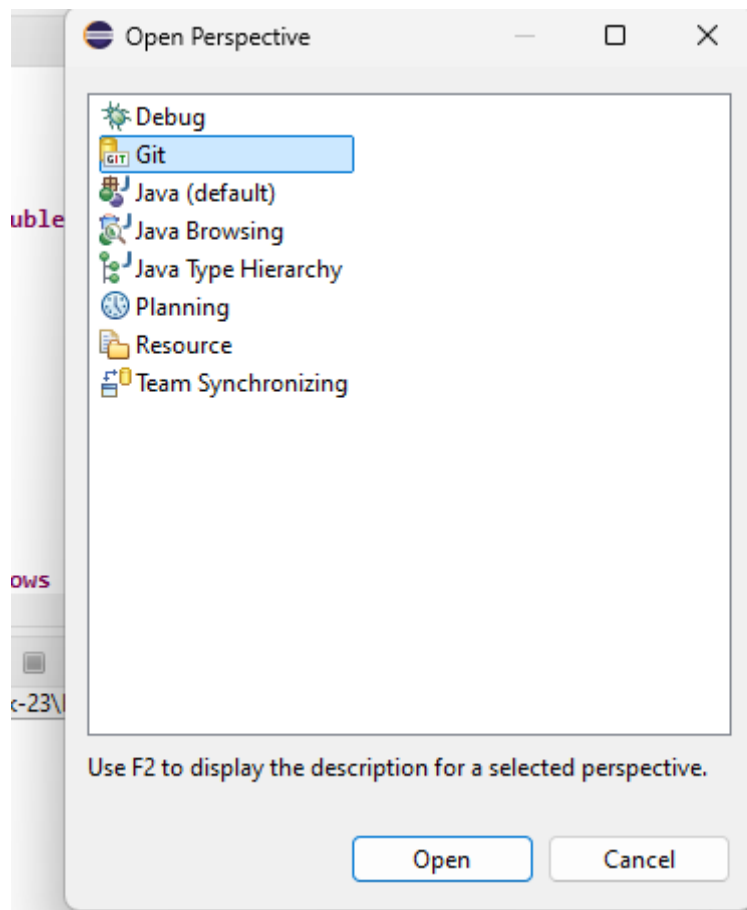
3.1. Inicializar Git y crear un repositorio

Configura Git en el proyecto e inicializa un repositorio público en GitHub

Inicializa un repositorio Git en tu proyecto

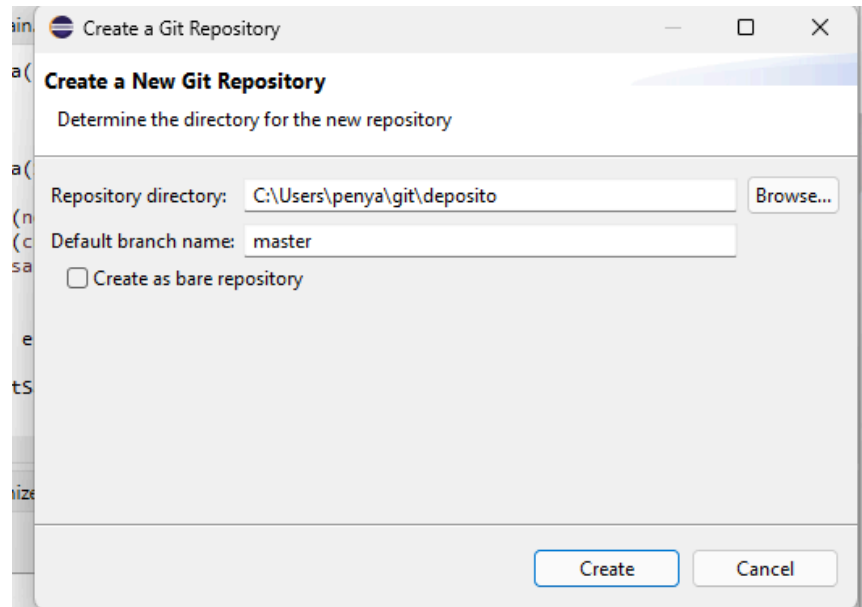
Cambiar a la perspectiva Git

1. **Cambiar a la perspectiva Git:** Ir al menú superior y seleccionar **Window > Perspective > Open Perspective > Other...**
2. En la ventana que aparece, seleccionar **Git** y hacer clic en **Open**.

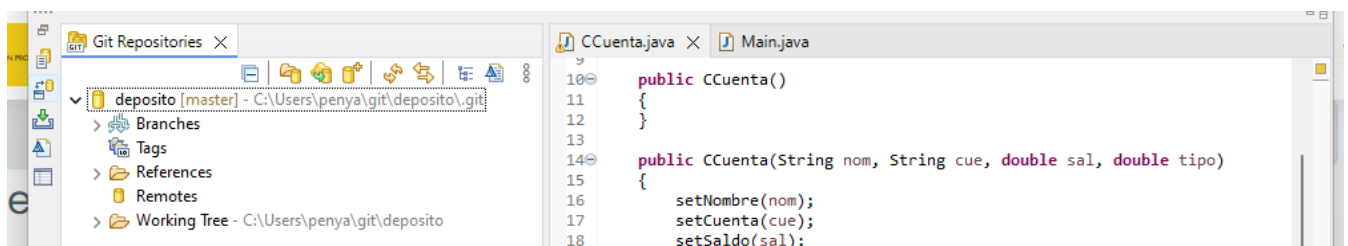


Crear un repositorio Git nuevo

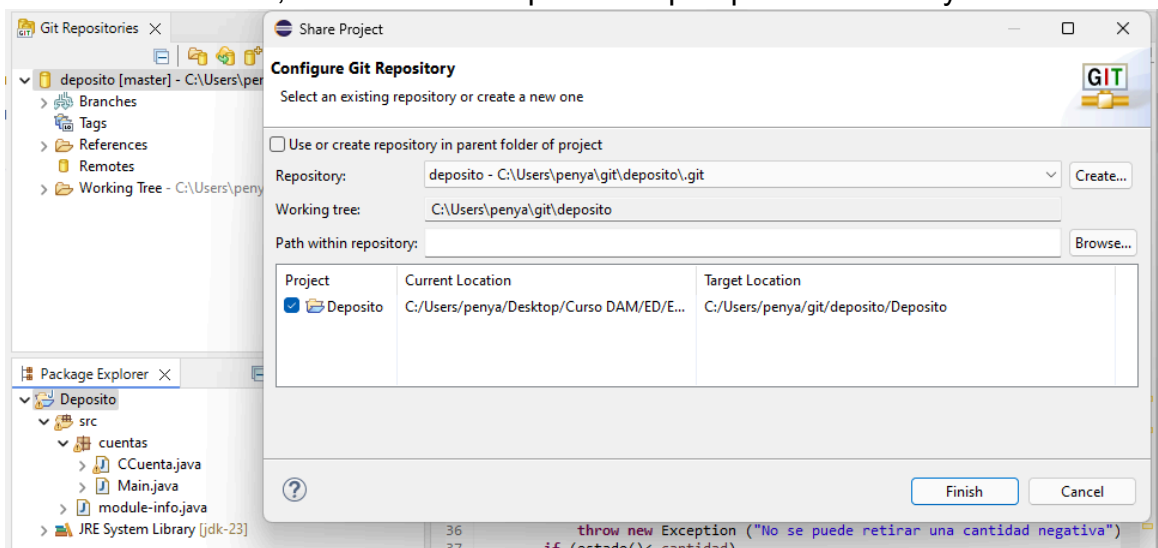
1. En la perspectiva **Git**, aparece el panel **Git Repositories** (también se puede activar desde **Window > Show View > Git Repositories**).
2. Hacer clic en el botón **Create a new Git Repository** (ícono con un signo "+" o "Add").
3. En la ventana que aparece: Seleccionar la carpeta raíz del proyecto local y hacer clic en **Create**.



Repositorio creado



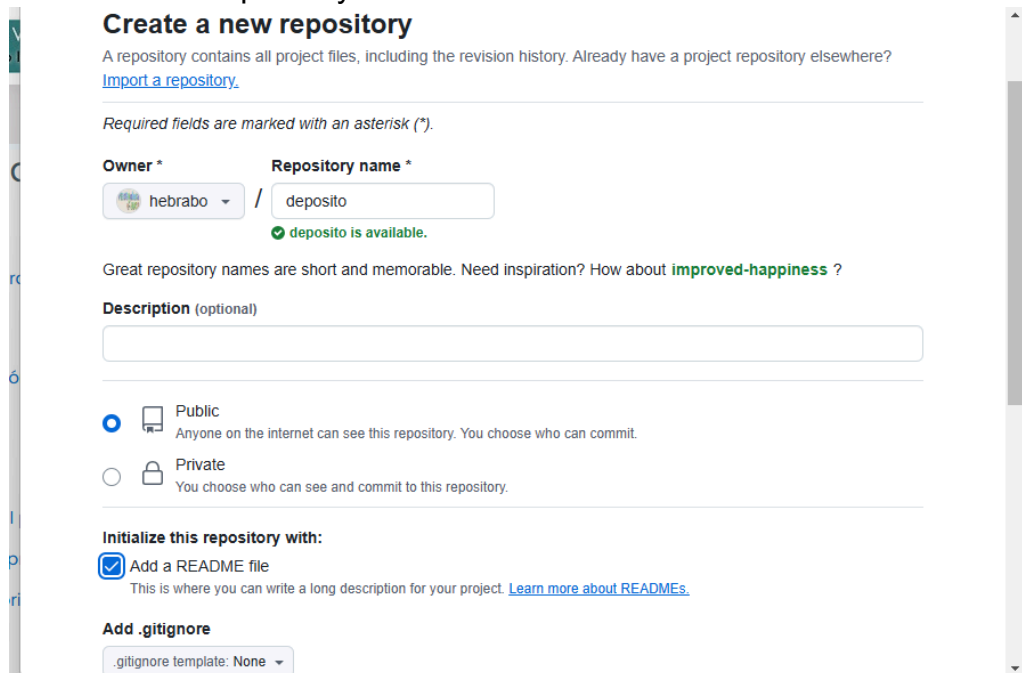
1. Desde la vista Package explorer, botón derecho sobre el proyecto / Teams / Share Project
2. En la nueva ventana, seleccionar el repositorio que queremos usar y seleccionar Finish.



Crea un repositorio público en GitHub y enlázalo al proyecto local

Crear un repositorio público en GitHub

1. Ir a GitHub (<https://github.com/>) y entrar con mi cuenta de usuario
2. Ir a mi perfil y seleccionar la pestaña Repositories
3. Seleccionar New para crear un nuevo repositorio
4. Poner nombre al repositorio (deposito)
5. Dejar el repositorio como Public
6. Seleccionar la casilla "Add README file"
7. Seleccionar "Create repository"



Create a new repository

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

Required fields are marked with an asterisk (*).

Owner * **Repository name ***

 hebrabo / deposito

✓ deposito is available.

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

Description (optional)

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

☐  **Private**
You choose who can see and commit to this repository.

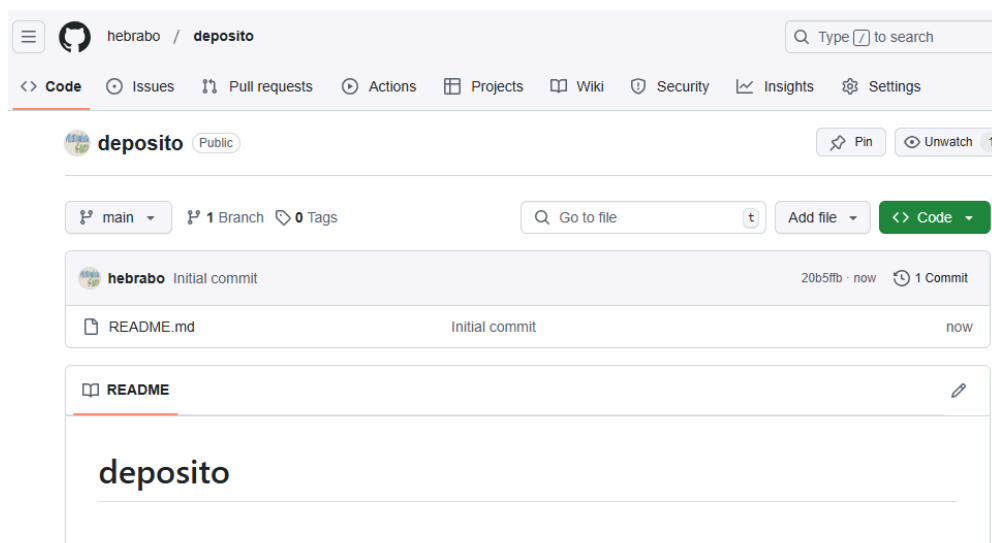
Initialize this repository with:

☒ **Add a README file**
This is where you can write a long description for your project. [Learn more about READMEs.](#)

Add .gitignore

.gitignore template: **None**

Repositorio creado en GitHub



hebrabo / deposito

Code Issues Pull requests Actions Projects Wiki Security Insights Settings

 **deposito** (Public) Pin Unwatch 1

main 1 Branch 0 Tags Go to file Add file Code

hebrabo Initial commit 20b5ffb · now 1 Commit

README.md Initial commit now

README

deposito

Generar un Token de acceso personal a GitHub

1. Ir a GitHub hacer clic en el avatar y seleccionar Settings.
2. En el menú de la izquierda, seleccionar Developer settings.
3. Luego, seleccionar Personal access tokens / Tokens y hacer clic en Generate new token.
4. Marcar los permisos que necesitamos (para este caso, marco solo los permisos de repo).
5. Hacer clic en Generate token al final de la página
6. Guardar el token en un lugar seguro, ya que no se puede ver de nuevo.

New personal access token (classic)

Personal access tokens (classic) function like ordinary OAuth access tokens. They can be used instead of a password for Git over HTTPS, or can be used to [authenticate to the API over Basic Authentication](#).

Note

Eclipse

What's this token for?

Expiration *

30 days ▾ The token will expire on Mon, Feb 24 2025

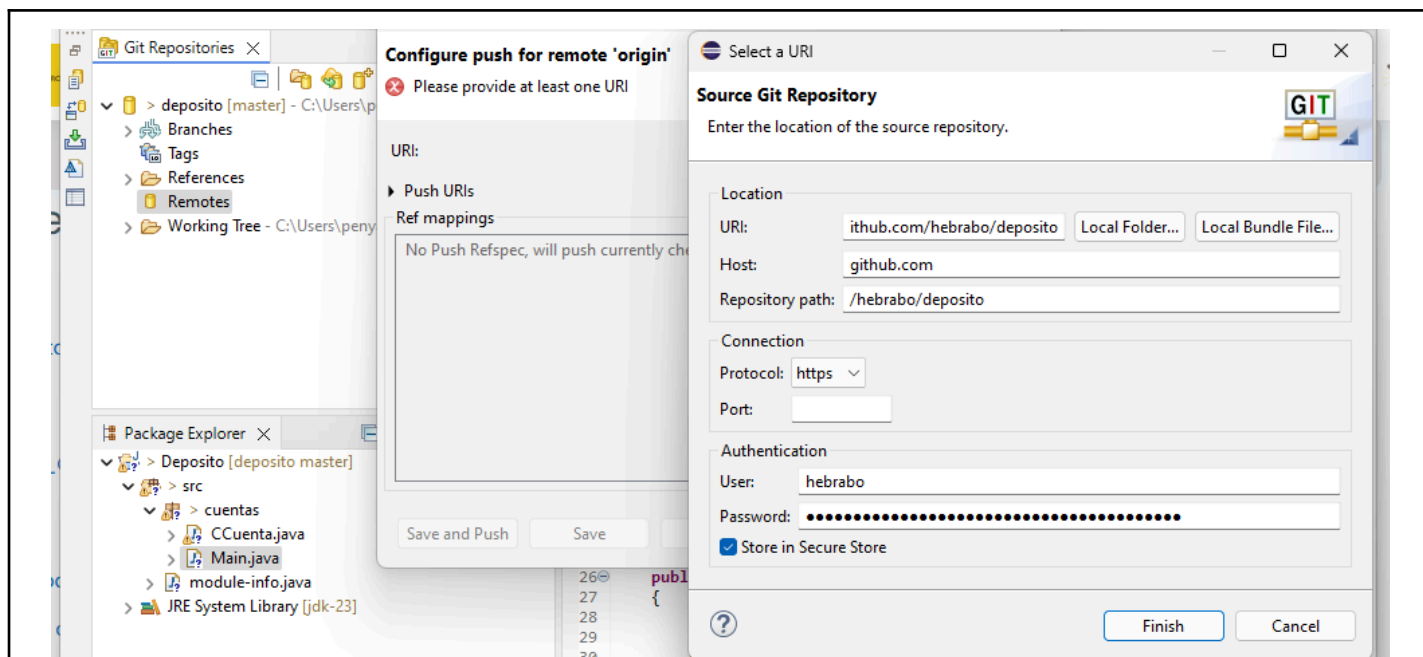
Select scopes

Scopes define the access for personal tokens. [Read more about OAuth scopes](#).

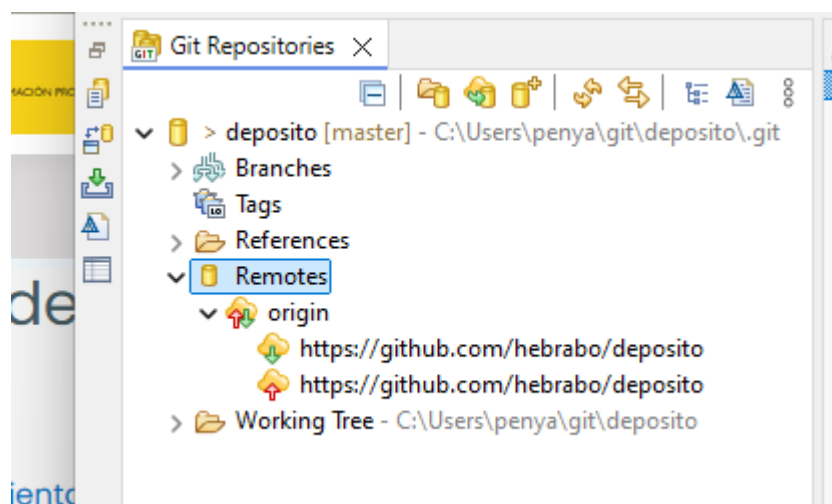
☒ repo	Full control of private repositories
☒ repo:status	Access commit status
☒ repo_deployment	Access deployment status
☒ public_repo	Access public repositories
☒ repo:invite	Access repository invitations
☒ security_events	Read and write security events

Enlazar el repositorio local de Eclipse con el repositorio público en GitHub

1. Copiar la URL del repositorio creado en GitHub (<https://github.com/hebrabo/deposito>)
2. Ir a Eclipse y en el panel Git Repositories, hacer clic derecho sobre el repositorio local y seleccionar "Remotes" > "New Remote"
3. En la nueva ventana dejar el nombre del Remote por defecto (origin), seleccionar Configure push y dar a Create.
4. En la nueva ventana, en **URI**, pegar la URL del repositorio que copiado de GitHub.
5. Escribir nuestro nombre de usuario y en password pegar el token generado en GitHub
6. Seleccionar la opción Store in secure store y clicar en Finish
7. Retornar a la ventana anterior donde seleccionamos Save.



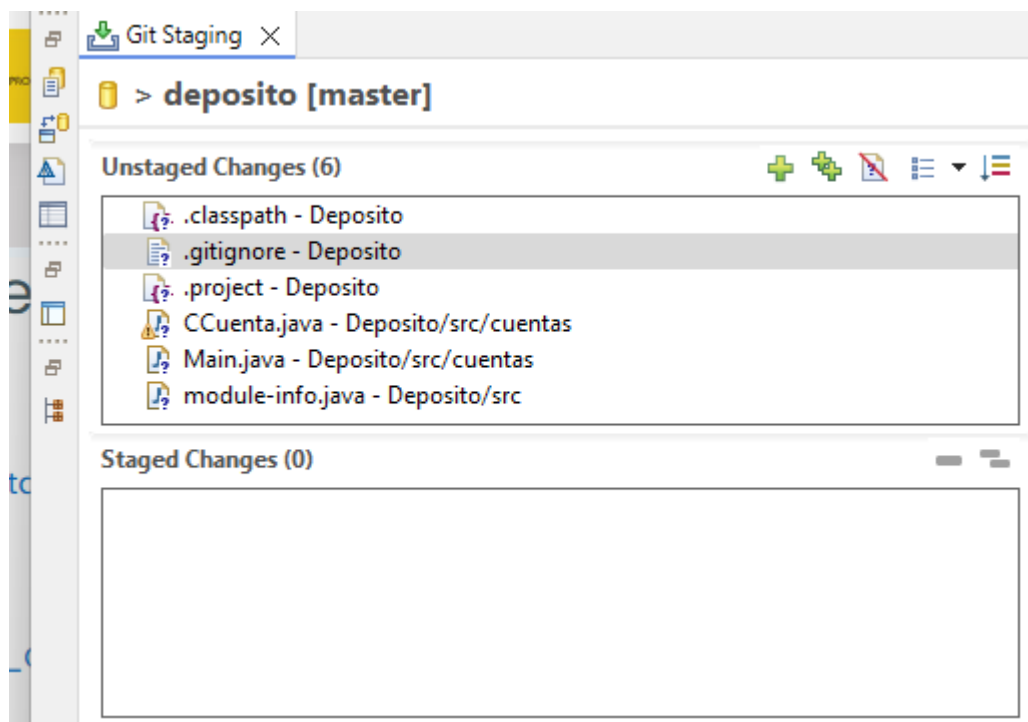
Enlace realizado



Realiza el primer commit

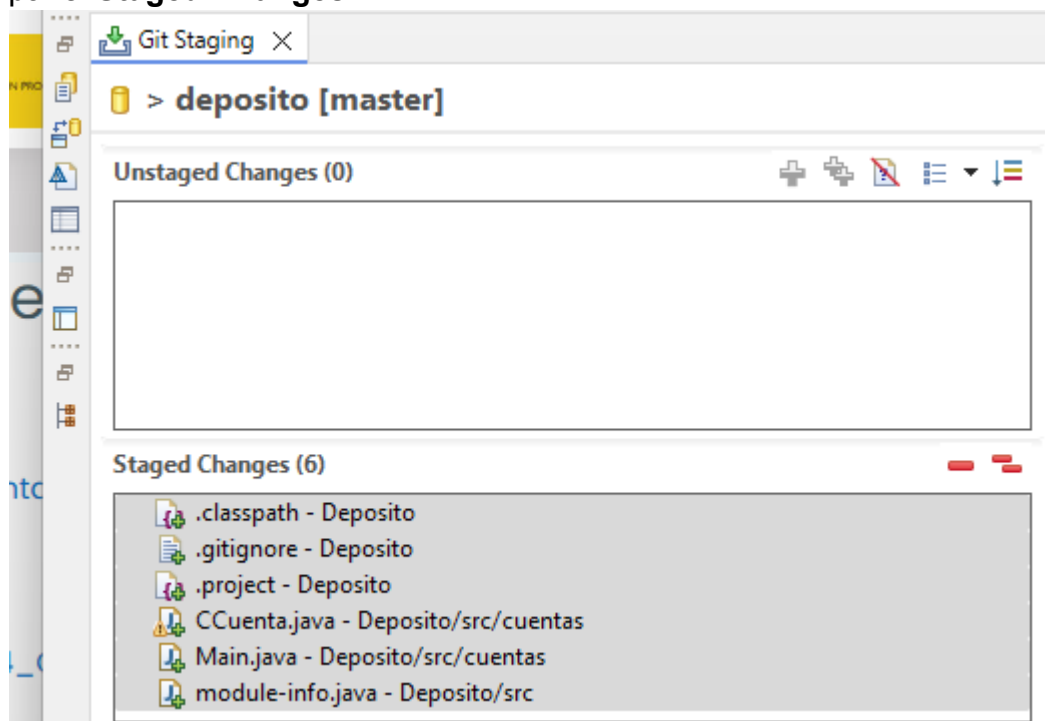
Ver los archivos que tenemos

1. En Eclipse abrir el panel Git Staging, yendo a **Window > Show View > Git Staging**
2. En el panel **Unstaged Changes** aparecen los archivos modificados o nuevos



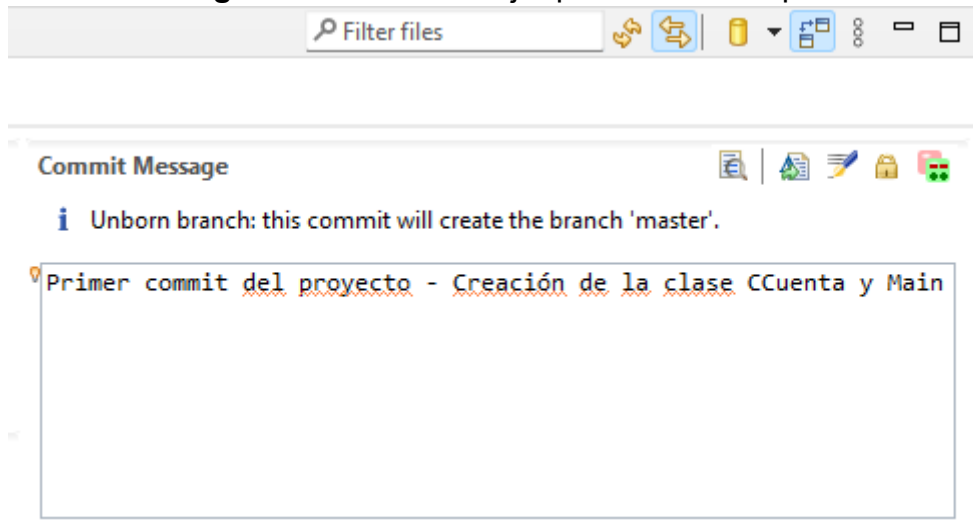
Añadir archivos al commit

Hacer clic en el icono **+** o seleccionar los archivos y hacer clic en el botón **Stage Changes** para moverlos al panel **Staged Changes**.



Escribir el mensaje de commit

En el campo **Commit Message** escribir un mensaje que describa lo que estamos realizando.



Realizar el commit

1. Hacer clic en el botón **Commit** para realizar el commit de los cambios (este botón realiza el commit solo en nuestro repositorio local, no sube nada a GitHub aún)
2. Abrir el Terminal y ejecutar el comando `git log` para ver los commits que hemos hecho



Explicaciones claras de lo que has hecho, por qué lo hiciste y qué lograste

¿Qué hemos hecho?	¿Por qué lo hemos hecho?	¿Qué hemos logrado?
Inicializamos Git y creamos un repositorio local.	Para comenzar a utilizar el control de versiones y gestionar los cambios en nuestro proyecto de manera estructurada.	Ahora tenemos un repositorio local que nos permite realizar commits y rastrear el historial de cambios del proyecto.

Creamos un repositorio público en GitHub.	Para compartir el proyecto con otros colaboradores y disponer de una copia remota accesible desde cualquier lugar.	Contamos con un repositorio remoto público donde podemos colaborar y sincronizar cambios con el repositorio local.
Enlazamos el repositorio local con el repositorio de GitHub.	Para conectar ambos repositorios y permitir el intercambio de cambios entre ellos mediante comandos como <code>push</code> y <code>pull</code> .	El repositorio local está vinculado al repositorio remoto, lo que facilita la colaboración y la gestión centralizada.
Realizamos el primer commit del proyecto.	Para registrar el estado inicial de los archivos y sentar la base del historial de versiones.	El repositorio local tiene un historial de commits iniciado, documentando los cambios realizados hasta ahora.
Abrimos el terminal para ver los commits realizados utilizando el comando <code>git log</code> .	Para verificar el historial de commits y asegurarnos de que los cambios se hayan registrado correctamente en el repositorio local.	Ahora podemos visualizar el historial de commits, incluyendo el mensaje del primer commit y su información asociada.

3.2. Realizar un commit descriptivo

Realiza al menos un commit con un mensaje descriptivo sobre los cambios realizados

Realiza modificaciones en el código

```

CCuenta.java  Main.java X
4-   public static void main(String[] args) {
5       operativa_cuenta(2500);
6   }
7
8-   public static void operativa_cuenta(float cantidad) {
9       CCuenta cuenta1;
10      double saldoActual;
11
12      cuenta1 = new CCuenta("Antonio López", "1000-2365-85-1230456789", cant
13      saldoActual = cuenta1.estado();
14      System.out.println("El saldo actual es de"+ saldoActual );
15
16      try {

```

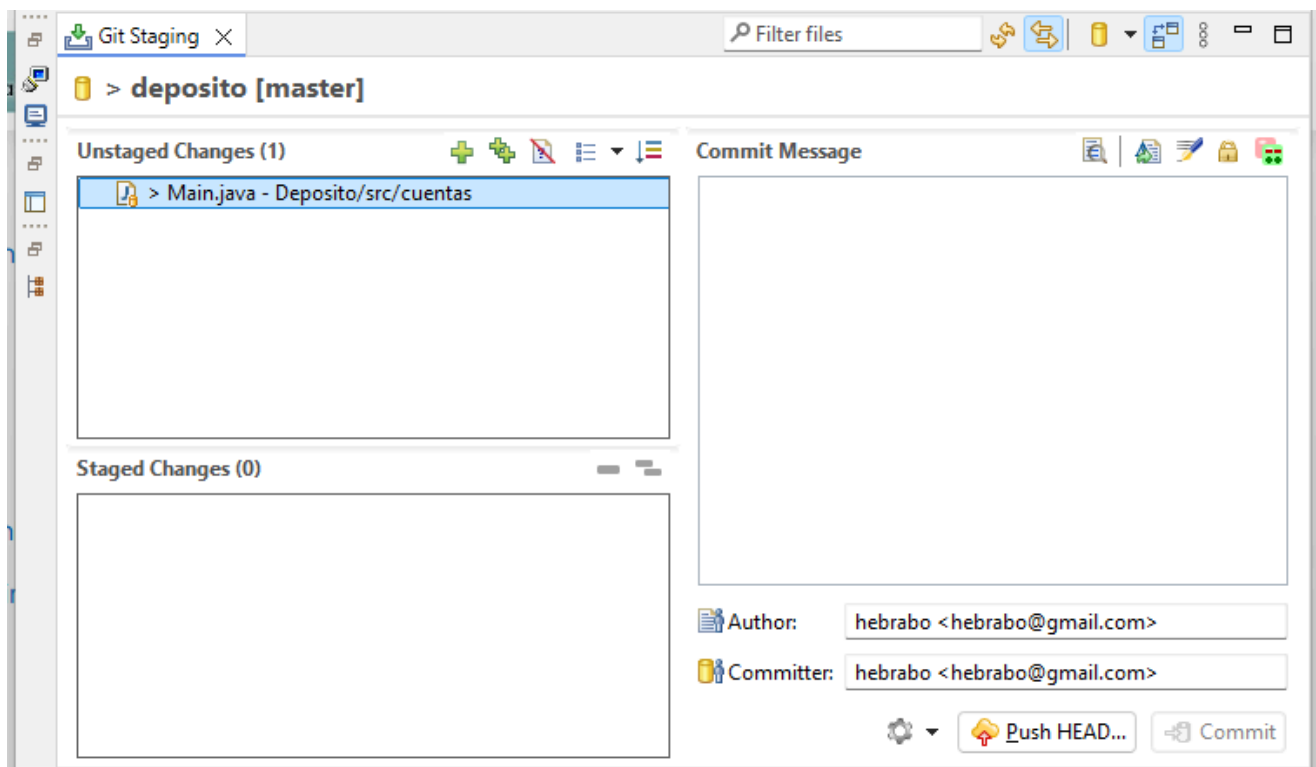
```

Terminal X Console
C:\WINDOWS\system32\cmd.exe X
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   cuentas/Main.java

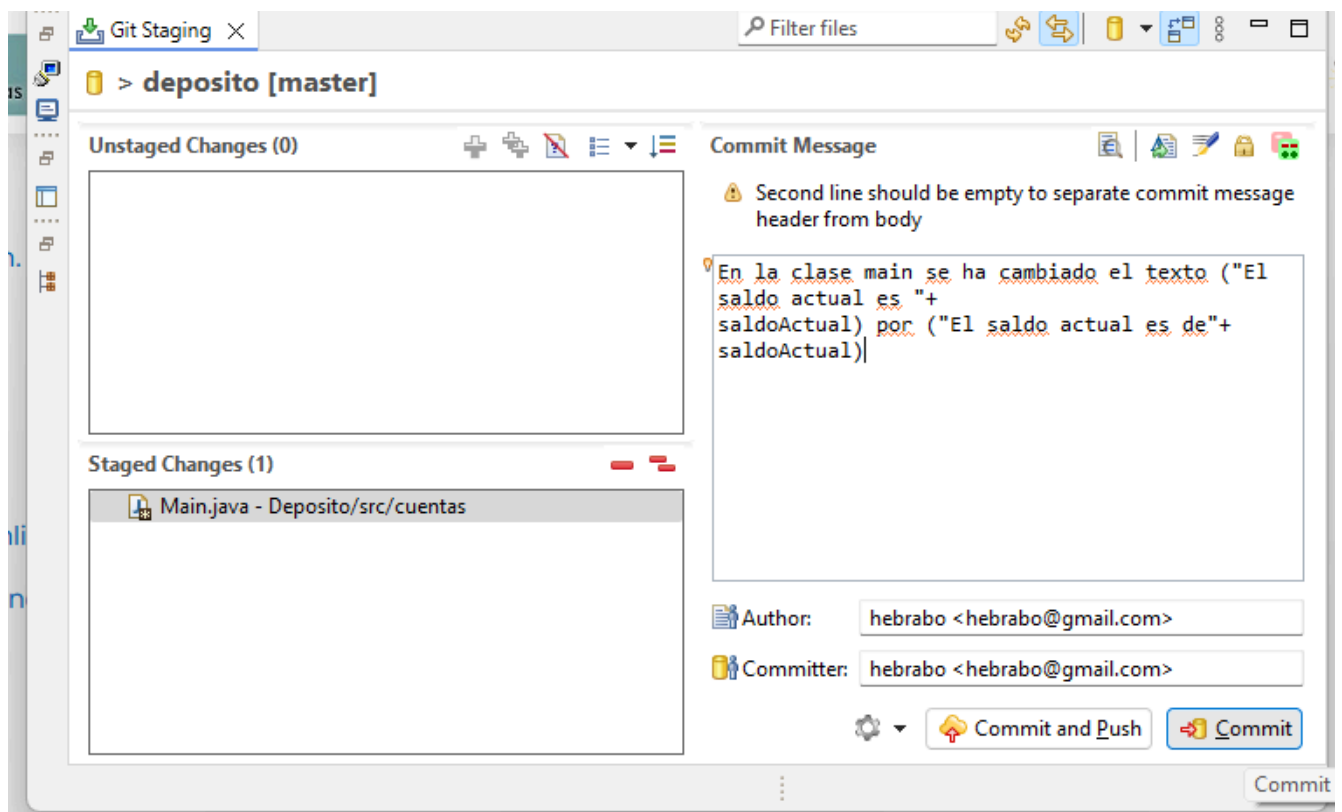
no changes added to commit (use "git add" and/or "git commit -a")

```

Añade los cambios al índice (git add)



Realiza el commit escribiendo un mensaje claro sobre los cambios realizados



Pantallazo del terminal mostrando el commit.

```

C:\WINDOWS\system32\cmd.exe - git log
(use "git restore <file>..." to discard changes in working directory)
modified:   cuentas/Main.java

no changes added to commit (use "git add" and/or "git commit -a")

C:\Users\penya\git\deposito\Deposito\src>git log
commit 9c52aa49a3084660a83d5233ef6461b5b38e1efb (HEAD -> master)
Author: hebrabo <hebrabo@gmail.com>
Date:   Wed Jan 29 09:01:59 2025 +0100

    En la clase main se ha cambiado el texto ("El saldo actual es "+
    saldoActual) por ("El saldo actual es de"+ saldoActual)
  
```

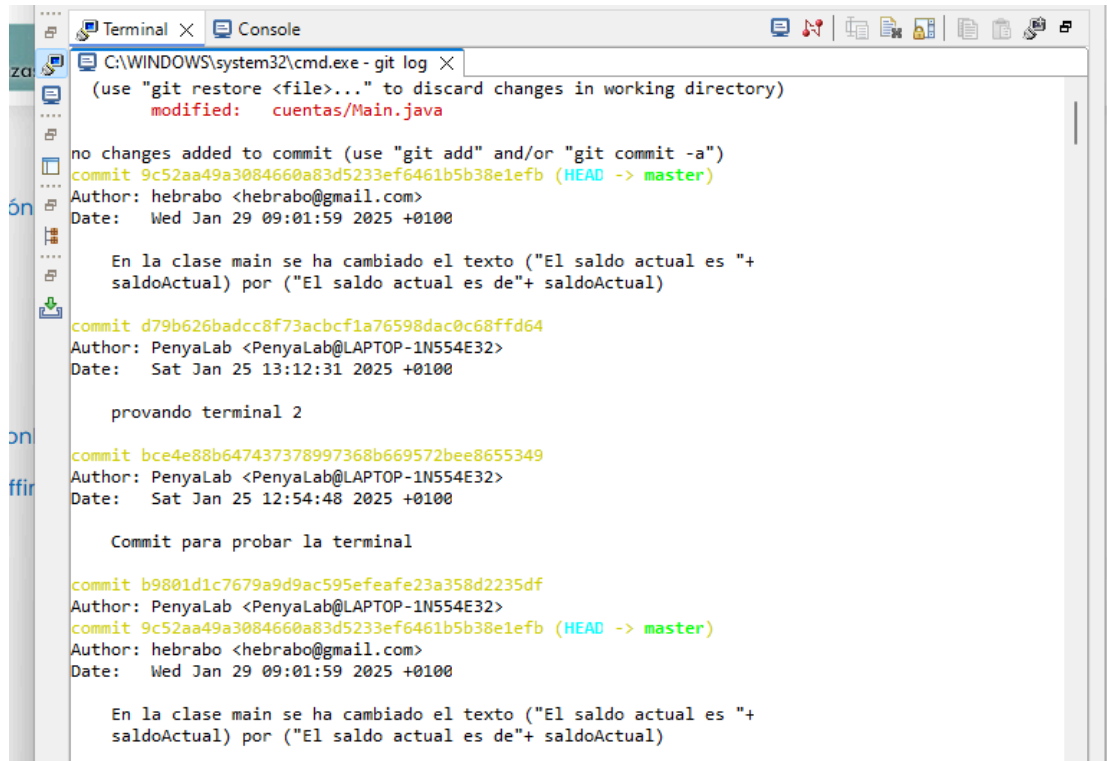
Explicaciones claras de lo que has hecho, por qué lo hiciste y qué lograste

¿Qué hemos hecho?	¿Por qué lo hemos hecho?	¿Qué hemos logrado?
Modificamos un archivo del proyecto	Lo hicimos para realizar algún cambio en el proyecto.	Se ha modificado el archivo según los requerimientos del proyecto.
Añadimos archivos modificados al índice.	Esto lo hicimos para preparar los archivos para el commit y asegurarnos de que solo los cambios deseados fueran registrados.	Los archivos modificados están ahora listos para ser incluidos en el commit.
Escribimos un mensaje claro en el commit para describir los cambios realizados.	El mensaje descriptivo es importante para que cualquier persona que vea el historial de commits entienda qué cambios se realizaron.	El commit tiene un mensaje claro y entendible, que facilita el seguimiento de cambios en el proyecto.
Hicimos el primer commit con un mensaje descriptivo.	Realizamos el commit para guardar el estado inicial del proyecto y dejar claro lo que hemos configurado.	El primer commit se ha realizado, registrando el estado actual del proyecto y permitiendo versiones futuras.

3.3. Mostrar el historial de versiones

Usa un comando en la consola para mostrar el historial de versiones del proyecto

Usa el comando git log en la consola



```
C:\WINDOWS\system32\cmd.exe - git log X
(use "git restore <file>..." to discard changes in working directory)
modified:   cuentas/Main.java

no changes added to commit (use "git add" and/or "git commit -a")
commit 9c52aa49a3084660a83d5233ef6461b5b38e1efb (HEAD -> master)
Author: hebrabo <hebrabo@gmail.com>
Date:   Wed Jan 29 09:01:59 2025 +0100

    En la clase main se ha cambiado el texto ("El saldo actual es "+
    saldoActual) por ("El saldo actual es de"+ saldoActual)

commit d79b626badcc8f73acbcf1a76598dac0c68ffd64
Author: Penyalab <Penyalab@LAPTOP-1N554E32>
Date:   Sat Jan 25 13:12:31 2025 +0100

    provando terminal 2

commit bce4e88b647437378997368b669572bee8655349
Author: Penyalab <Penyalab@LAPTOP-1N554E32>
Date:   Sat Jan 25 12:54:48 2025 +0100

    Commit para probar la terminal

commit b9801d1c7679a9d9ac595efeafe23a358d2235df
Author: Penyalab <Penyalab@LAPTOP-1N554E32>
commit 9c52aa49a3084660a83d5233ef6461b5b38e1efb (HEAD -> master)
Author: hebrabo <hebrabo@gmail.com>
Date:   Wed Jan 29 09:01:59 2025 +0100

    En la clase main se ha cambiado el texto ("El saldo actual es "+
    saldoActual) por ("El saldo actual es de"+ saldoActual)
```

Explicaciones claras de lo que has hecho, por qué lo hiciste y qué lograste

¿Qué hemos hecho?	¿Por qué lo hemos hecho?	¿Qué hemos logrado?
Usamos <code>git log</code> para ver el historial de commits	Para revisar los commits anteriores y entender el progreso del proyecto o la historia de los cambios realizados.	Hemos obtenido el historial completo de los commits del proyecto, lo que nos permite visualizar cómo ha evolucionado.

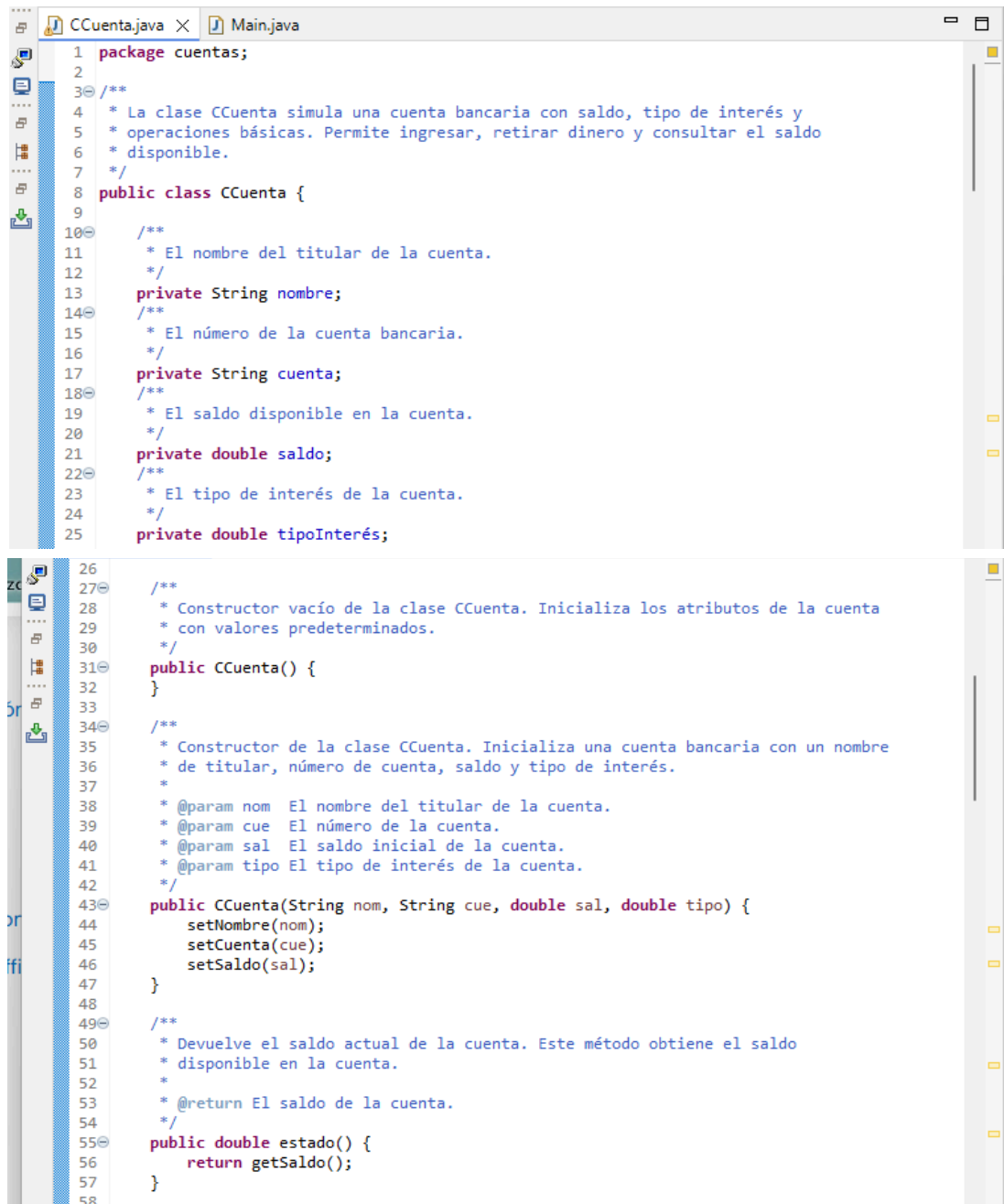
Ejercicio 4: Documentación con Javadoc

4.1. Insertar comentarios Javadoc

Fragmento del código mostrando los comentarios añadidos

Añade comentarios Javadoc a todos los métodos y atributos de la clase CCuenta

Añade comentarios Javadoc sobre el propósito de cada método y atributo



```
1 package cuentas;
2
3 /**
4  * La clase CCuenta simula una cuenta bancaria con saldo, tipo de interés y
5  * operaciones básicas. Permite ingresar, retirar dinero y consultar el saldo
6  * disponible.
7  */
8 public class CCuenta {
9
10     /**
11      * El nombre del titular de la cuenta.
12      */
13     private String nombre;
14     /**
15      * El número de la cuenta bancaria.
16      */
17     private String cuenta;
18     /**
19      * El saldo disponible en la cuenta.
20      */
21     private double saldo;
22     /**
23      * El tipo de interés de la cuenta.
24      */
25     private double tipoInterés;
26
27     /**
28      * Constructor vacío de la clase CCuenta. Inicializa los atributos de la cuenta
29      * con valores predeterminados.
30      */
31     public CCuenta() {
32     }
33
34     /**
35      * Constructor de la clase CCuenta. Inicializa una cuenta bancaria con un nombre
36      * de titular, número de cuenta, saldo y tipo de interés.
37      *
38      * @param nom El nombre del titular de la cuenta.
39      * @param cue El número de la cuenta.
40      * @param sal El saldo inicial de la cuenta.
41      * @param tipo El tipo de interés de la cuenta.
42      */
43     public CCuenta(String nom, String cue, double sal, double tipo) {
44         setNombre(nom);
45         setCuenta(cue);
46         setSaldo(sal);
47     }
48
49     /**
50      * Devuelve el saldo actual de la cuenta. Este método obtiene el saldo
51      * disponible en la cuenta.
52      *
53      * @return El saldo de la cuenta.
54      */
55     public double estado() {
56         return getSaldo();
57     }
58 }
```

```

58
59- /**
60  * Ingresa una cantidad de dinero en la cuenta. Si la cantidad es negativa, se
61  * lanza una excepción.
62  *
63  * @param cantidad La cantidad de dinero que se desea ingresar en la cuenta.
64  * @throws Exception Si la cantidad es negativa.
65  */
66- public void ingresar(double cantidad) throws Exception {
67     if (cantidad < 0)
68         throw new Exception("No se puede ingresar una cantidad negativa");
69     setSaldo(getSaldo() + cantidad);
70 }
71
72- /**
73  * Retira una cantidad de dinero de la cuenta. Si la cantidad es negativa o el
74  * saldo es insuficiente, se lanza una excepción.
75  *
76  * @param cantidad La cantidad de dinero que se desea retirar de la cuenta.
77  * @throws Exception Si la cantidad es negativa o si no hay suficiente saldo.
78  */
79- public void retirar(double cantidad) throws Exception {
80     if (cantidad <= 0)
81         throw new Exception("No se puede retirar una cantidad negativa");
82     if (estado() < cantidad)
83         throw new Exception("No se hay suficiente saldo");
84     setSaldo(getSaldo() - cantidad);
85 }
86

```

```

86
87- /**
88  * Obtiene el tipo de interés de la cuenta.
89  *
90  * @return El tipo de interés de la cuenta.
91  */
92- private double getTipoInterés() {
93     return tipoInterés;
94 }
95
96- /**
97  * Establece el tipo de interés de la cuenta.
98  *
99  * @param tipoInterés El tipo de interés a establecer.
100 */
101- private void setTipoInterés(double tipoInterés) {
102     this.tipoInterés = tipoInterés;
103 }
104
105- /**
106  * Obtiene el saldo actual de la cuenta.
107  *
108  * @return El saldo de la cuenta.
109  */
110- private double getSaldo() {
111     return saldo;
112 }
113

```

```

113
114- /**
115  * Establece el saldo de la cuenta.
116  *
117  * @param saldo El saldo a establecer en la cuenta.
118  */
119- private void setSaldo(double saldo) {
120     this.saldo = saldo;
121 }
122
123- /**
124  * Obtiene el número de la cuenta.
125  *
126  * @return El número de la cuenta.
127  */
128- private String getCuenta() {
129     return cuenta;
130 }
131
132- /**
133  * Establece el número de la cuenta.
134  *
135  * @param cuenta El número de la cuenta a establecer.
136  */
137- private void setCuenta(String cuenta) {
138     this.cuenta = cuenta;
139 }
140

```

```

141  /**
142   * Obtiene el nombre del titular de la cuenta.
143   *
144   * @return El nombre del titular de la cuenta.
145   */
146  private String getNombre() {
147      return nombre;
148  }
149
150  /**
151   * Establece el nombre del titular de la cuenta.
152   *
153   * @param nombre El nombre del titular de la cuenta.
154   */
155  private void setNombre(String nombre) {
156      this.nombre = nombre;
157  }
158  }

```



Explicaciones claras de lo que has hecho, por qué lo hiciste y qué lograste

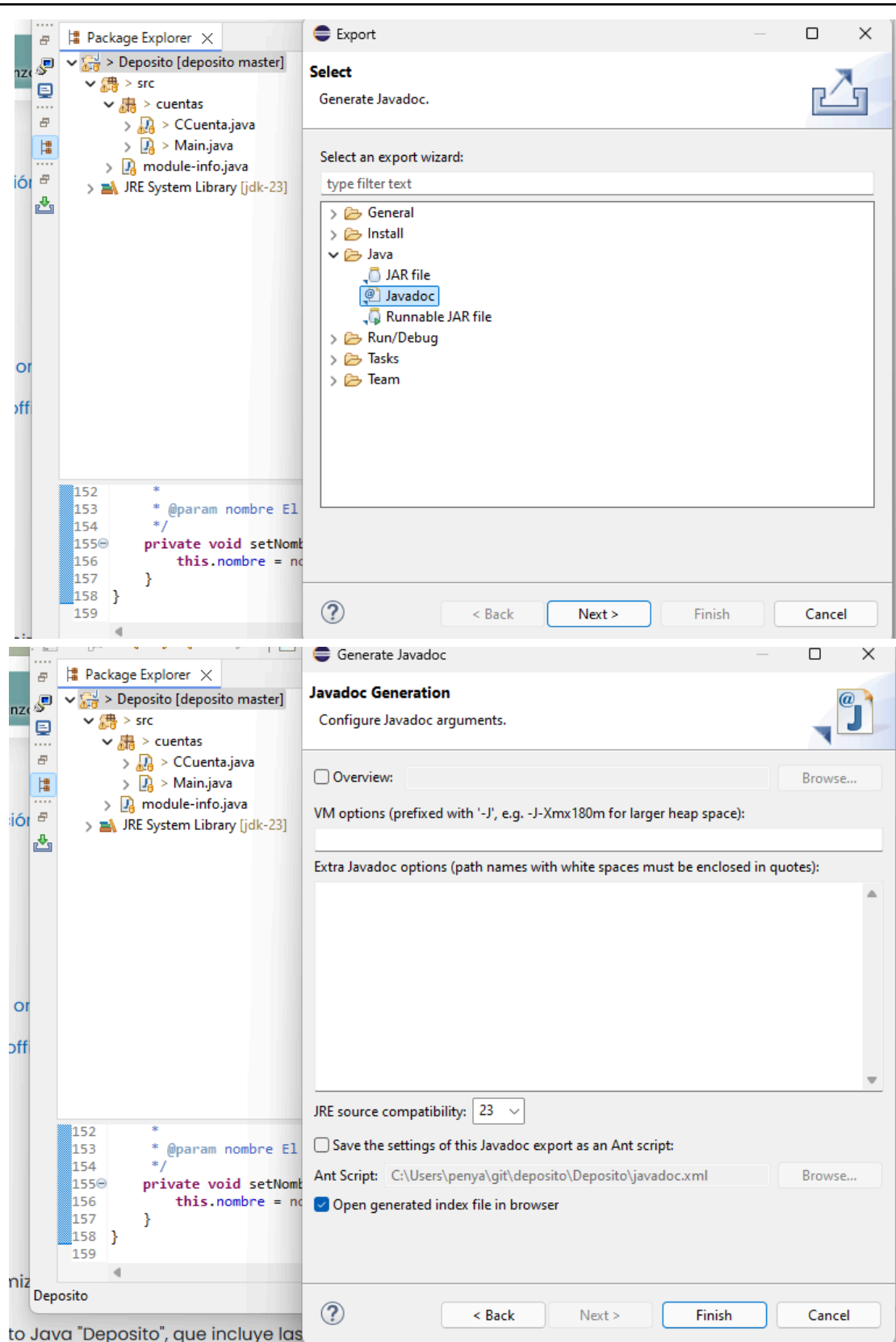
¿Qué hemos hecho?	¿Por qué lo hemos hecho?	¿Qué hemos logrado?
Hemos añadido comentarios Javadoc a los métodos y atributos de la clase Main y de la clase	Lo hicimos para documentar claramente cada método y parámetro en el código, ayudando a que otros desarrolladores entiendan la funcionalidad del código de forma rápida y eficiente.	Ahora el código está correctamente documentado con Javadoc, lo que facilita la comprensión del propósito y funcionamiento de cada parte del código. Esto también permite generar documentación de forma automática usando herramientas como Javadoc.

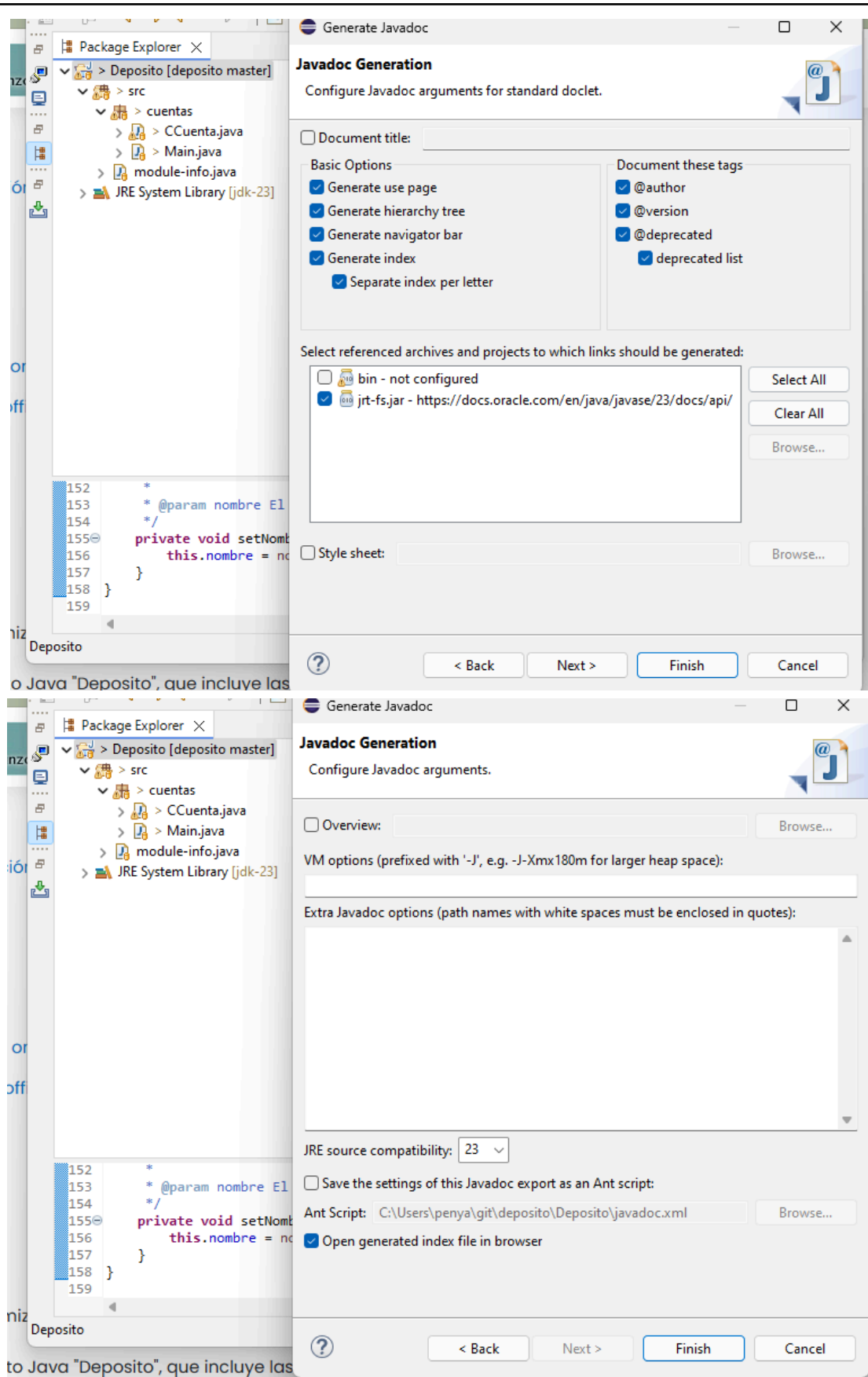
4.2. Generar documentación Javadoc

Genera la documentación Javadoc para todo el proyecto

Usa las herramientas del IDE para generar la documentación Javadoc

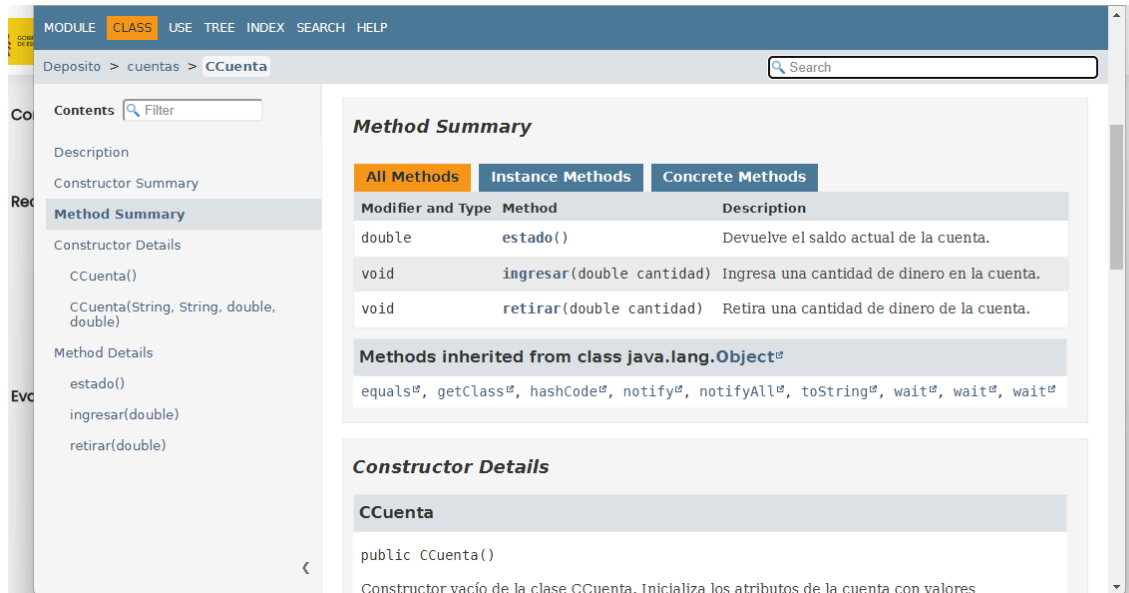
1. En la **vista de Project Explorer**, hacer clic derecho sobre el nombre del proyecto.
2. En el menú contextual, seleccionar **Export**.
3. En la ventana emergente, seleccionar **Java > Javadoc**.
4. Hacer clic en **Next**.
5. En las siguientes ventanas configurar las opciones de exportación
6. Hacer clic en **Finish**





Abre la documentación generada en un navegador

1. Ir al directorio donde se ha guardado
2. Buscar el archivo **index.html** y hacer doble clic para abrirlo en el navegador.



Explicaciones claras de lo que has hecho, por qué lo hiciste y qué lograste

¿Qué hemos hecho?	¿Por qué lo hemos hecho?	¿Qué hemos logrado?
Generamos la documentación Javadoc utilizando Eclipse.	Para crear una documentación HTML a partir de los comentarios Javadoc, lo que facilita la navegación y la comprensión del proyecto de manera visual.	Se generó un conjunto de archivos HTML con la documentación Javadoc, accesibles a través de un navegador.
Abrimos el archivo <code>index.html</code> generado para ver la documentación.	Para revisar que la documentación generada esté correcta y completa, y verificar que se incluyan todas las clases y métodos documentados.	Acceso a la documentación del proyecto en formato HTML, lo que permite consultar fácilmente la descripción de las clases y métodos desde un navegador.

Conclusión

Breve reflexión sobre los aprendizajes obtenidos

A lo largo de este ejercicio, he adquirido una comprensión más profunda y práctica sobre varios aspectos clave en el desarrollo de software, especialmente en el manejo de **Git**, **GitHub** y la **documentación con Javadoc**. Aquí están los aprendizajes más significativos:

1. Refactorización de código:

- El proceso de mover las clases a un paquete y renombrar variables me permitió aplicar buenas prácticas de organización de código, lo que facilita la escalabilidad y el mantenimiento del proyecto.
- La creación del método `operativa_cuenta` me ayudó a comprender la importancia de **modularizar** el código, es decir, dividir el código en funciones/métodos reutilizables para mejorar la legibilidad y la eficiencia.
- El uso de **encapsulación** en la clase `CCuenta` me permitió entender cómo proteger los datos internos de una clase y cómo usar métodos getter y setter para interactuar con esos datos de forma controlada.

2. Uso de Git y GitHub:

- La inicialización de un repositorio Git y su integración con GitHub me permitió gestionar el código de manera más eficiente, realizando commits y manteniendo un historial claro de las modificaciones.
- Aprendí cómo usar los comandos de Git para realizar commits, ver el historial de versiones y colaborar en proyectos utilizando un repositorio remoto.
- Este ejercicio me enseñó la importancia de tener un control de versiones en proyectos, permitiendo la colaboración en equipo, el seguimiento de cambios y la recuperación de versiones anteriores si es necesario.

3. Documentación Javadoc:

- Añadir comentarios Javadoc me permitió comprender cómo documentar el propósito de las clases, métodos y atributos de manera efectiva.
- La generación de la documentación Javadoc me permitió ver cómo crear documentación técnica accesible y útil para otros desarrolladores que trabajen en el proyecto, lo que hace que el proyecto sea más mantenible y entendible a largo plazo.
- A través de la documentación, aprendí cómo describir detalladamente los elementos de código y cómo generar un archivo HTML que sirve como guía para comprender el funcionamiento del sistema.