



# **Tarea de Evaluación 3t**

Alejandro Garcia-Mauriño  
Salas . Antonio Moro del Toro

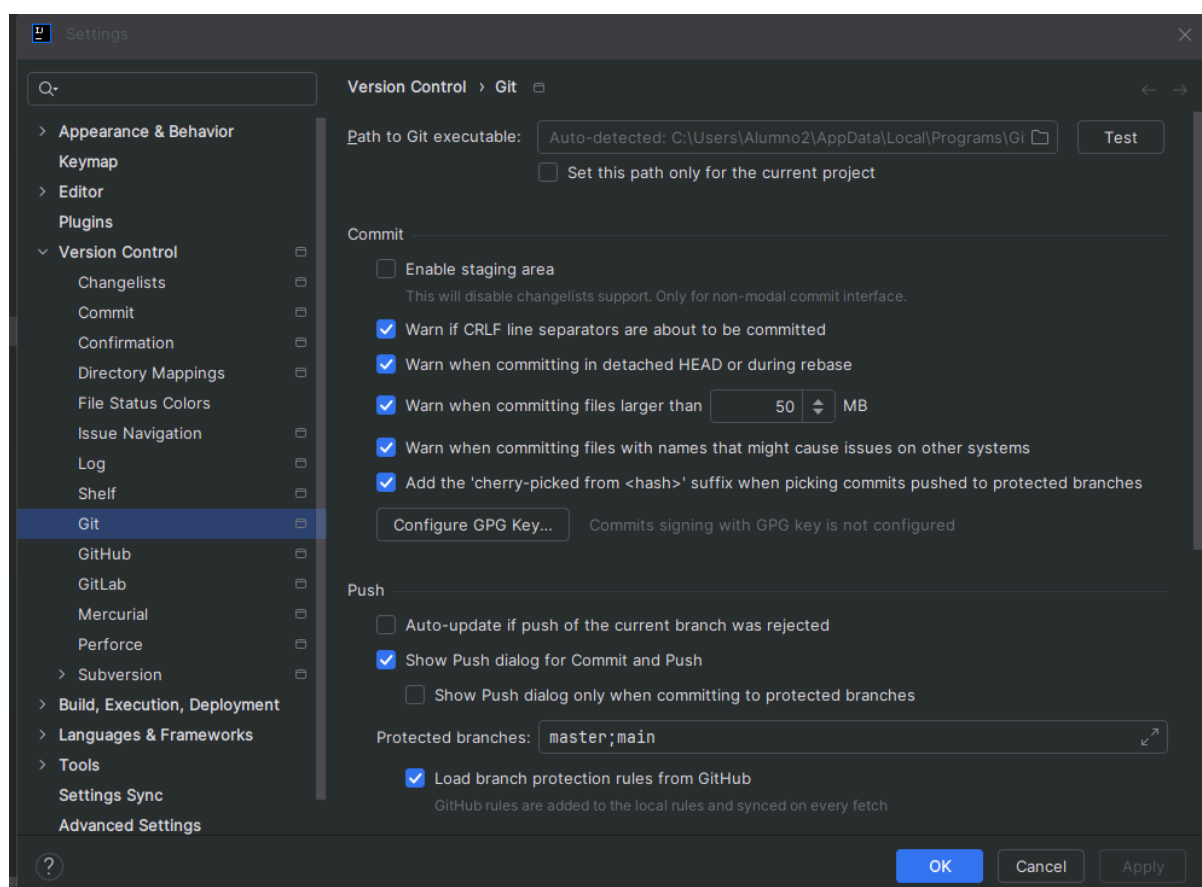


# 1. Configurar Git y GitHub en IntelliJ IDEA

Git te ayuda a llevar un historial de cambios en tu código, y GitHub permite compartir y colaborar en proyectos.

## Activar Git en tu proyecto

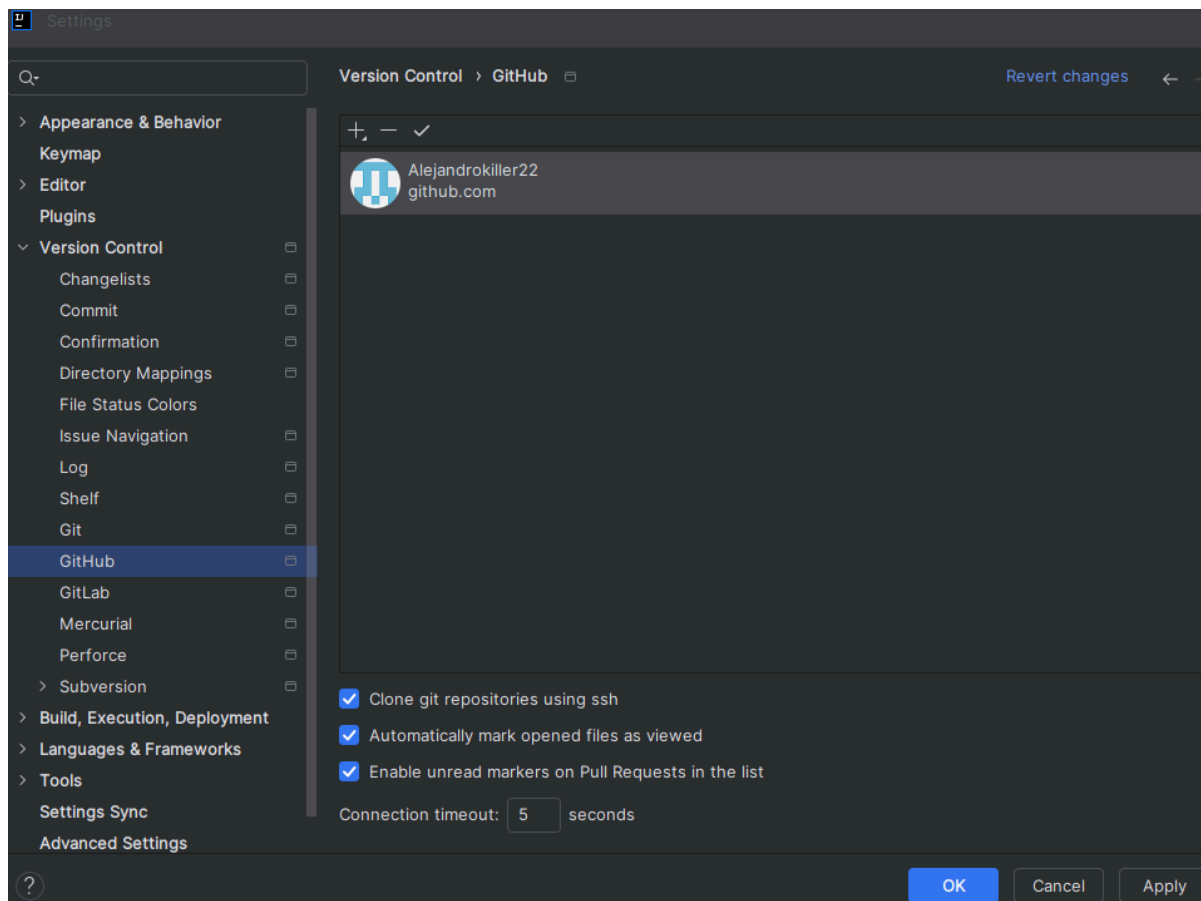
**Abre tu proyecto en IntelliJ IDEA.** Ve a **VCS > Enable Version Control Integration...** Selecciona **Git** y presiona **OK**. Se creará la carpeta `.git`, lo que indica que tu proyecto está bajo control de versiones.



## Conectar GitHub a IntelliJ



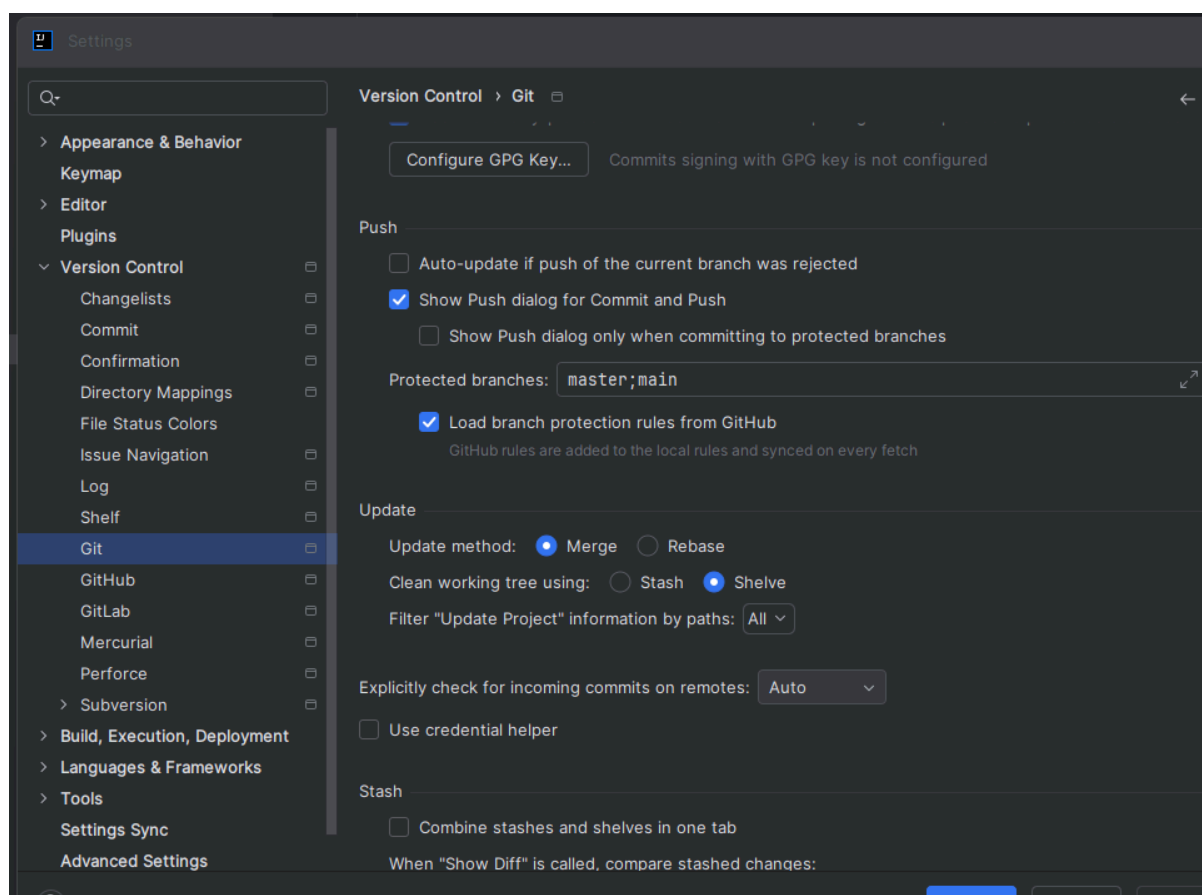
Ve a **File > Settings > Version Control > GitHub**. Presiona + **Add Account** y selecciona **Log in via GitHub**. **Inicia sesión con tu cuenta** y autoriza IntelliJ.



## Subir tu código a GitHub

Ve a **VCS > Git > Push**. Si no tienes un repositorio remoto, ve a **GitHub > New Repository** y crea uno. Copia la URL y usa el comando en la terminal de IntelliJ:

```
git remote add origin https://github.com/TuUsuario/NombreRepositorio.git
```



Usa `git push origin main` para subir tus archivos a GitHub.

**El proyecto ahora está en GitHub y bajo control de versiones.**

## Commits frecuentes y descriptivos

### Revisar los cambios

- Asegúrate de que las modificaciones en el archivo son correctas y necesarias para el proyecto.
- En este caso, se han agregado configuraciones relacionadas con **JUnit** (bibliotecas y documentación).

### 2. Añadir un mensaje descriptivo para el commit



En el cuadro de texto, escribe un mensaje claro sobre qué estás cambiando, por ejemplo:

Configuración de bibliotecas JUnit en IntelliJ IDEA

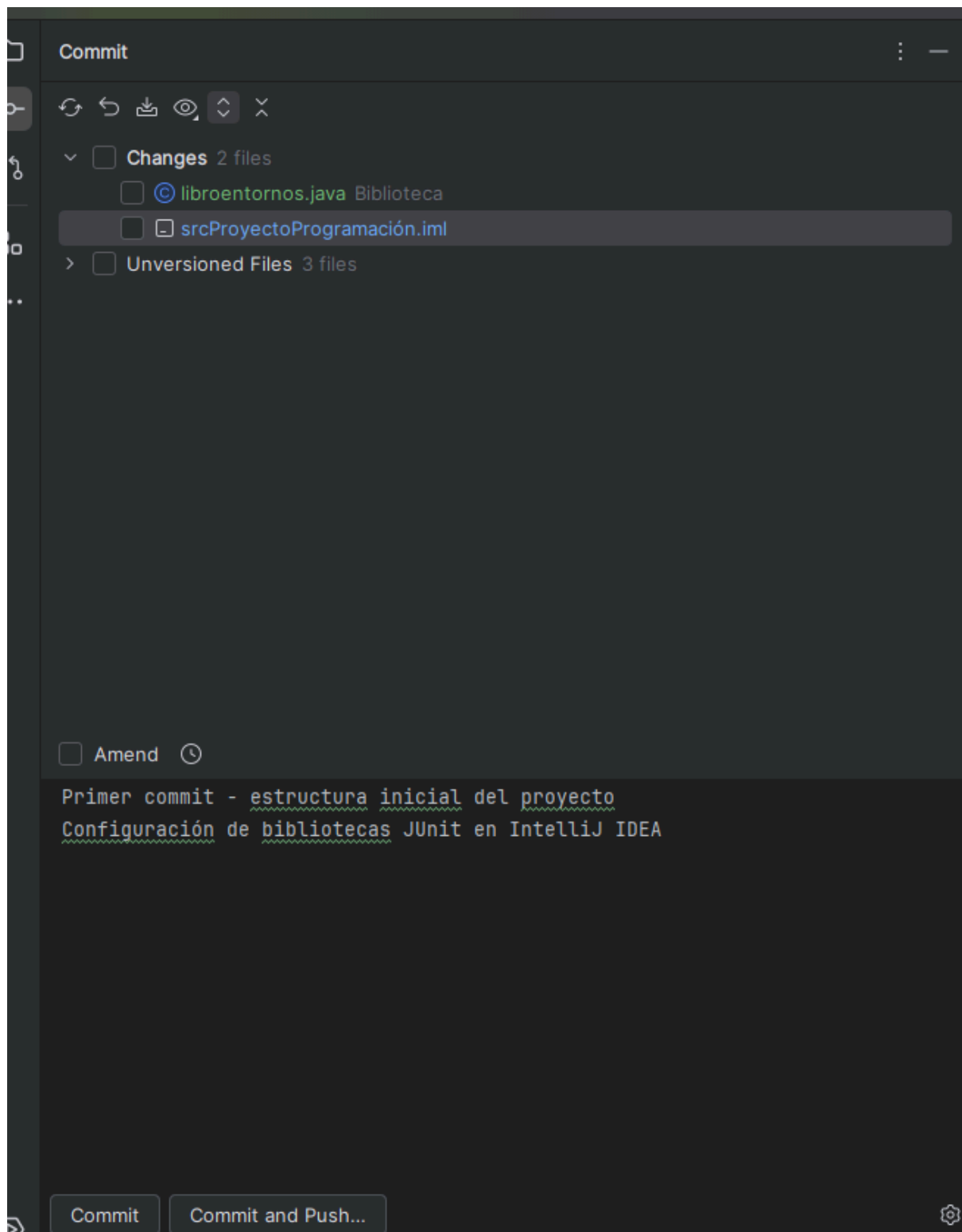
- Un mensaje claro ayuda a entender el propósito del cambio en el historial de versiones.

### 3 Realizar el commit

- Haz clic en **Commit** para guardar los cambios localmente.
- Si deseas subir los cambios a GitHub, haz **Commit and Push**.

### 4 Verificar en GitHub

- Abre tu repositorio en GitHub y revisa si los cambios aparecen correctamente después del **push**.





↑ ↓ ↻ ↶ ↷

Side-by-side viewer

Do not ignore

Highlight words

✕ ↕ ⚙ ?

cf53fad40223b0881f3640caee12140c07493ac5

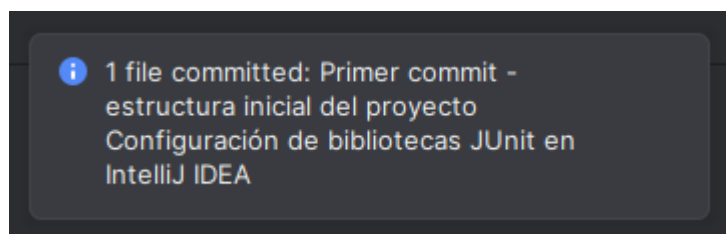
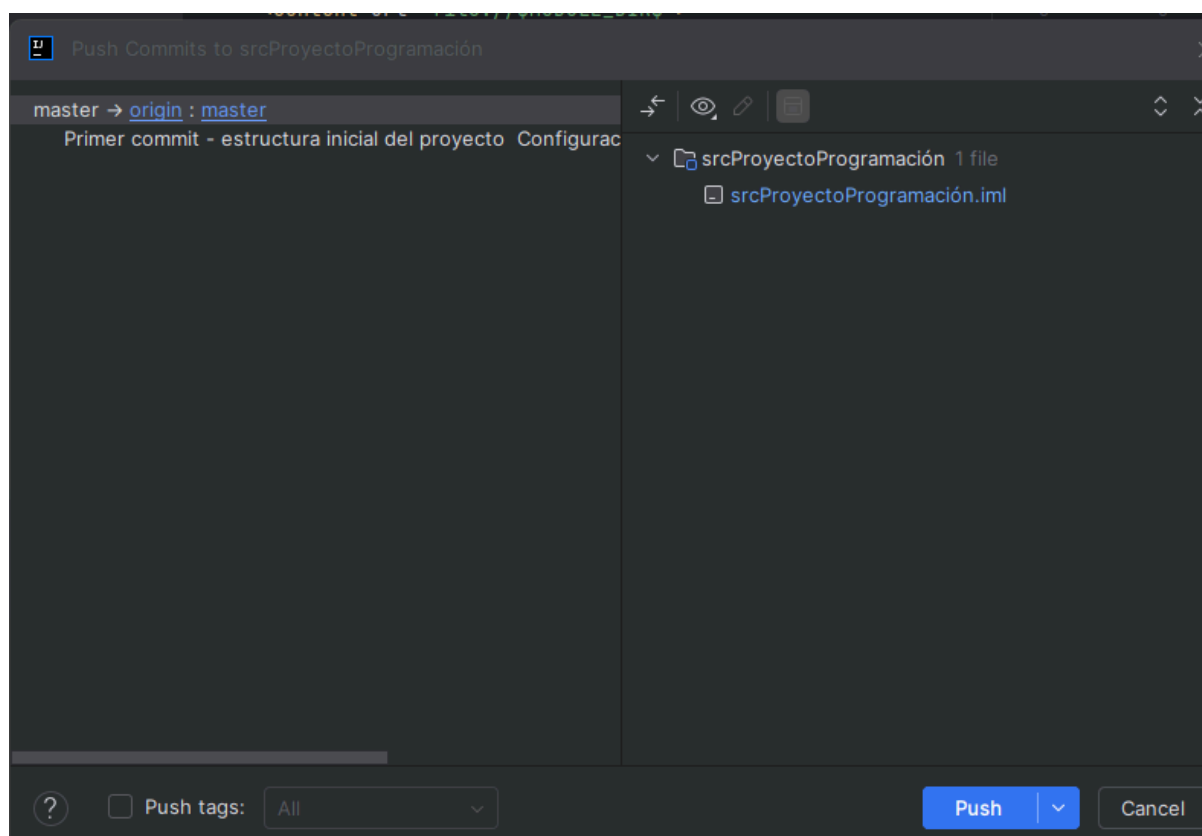
<?xml version="1.0" encoding="UTF-8"?>  
<module type="JAVA\_MODULE" version="4">  
 <component name="NewModuleRootManager" inherit-compiler-output  
 <exclude-output />  
 <content url="file://\$MODULE\_DIR\$">  
 <sourceFolder url="file://\$MODULE\_DIR\$" isTestSource="false" />  
 </content>  
 <orderEntry type="inheritedJdk" />  
 <orderEntry type="sourceFolder" forTests="false" />  
 </component>  
</module>

>> 10  
11

☐ Your version

<?xml version="1.0" encoding="UTF-8"?>  
<module type="JAVA\_MODULE" version="4">  
 <component name="NewModuleRootManager" inherit-compiler-output  
 <exclude-output />  
 <content url="file://\$MODULE\_DIR\$">  
 <sourceFolder url="file://\$MODULE\_DIR\$" isTestSource="false" />  
 </content>  
 <orderEntry type="inheritedJdk" />  
 <orderEntry type="sourceFolder" forTests="false" />  
 </component>  
</module>

10  
11  
12  
13  
14  
15  
16  
17  
18  
19  
20  
21  
22  
23  
24  
25  
26  
27  
28  
29



## Configurar JUnit manualmente en IntelliJ IDEA

### 1 Descargar JUnit

- Ve a la página oficial de **JUnit**: <https://junit.org/junit4/>
- Descarga el archivo `junit-4.13.2.jar` (versión más reciente).

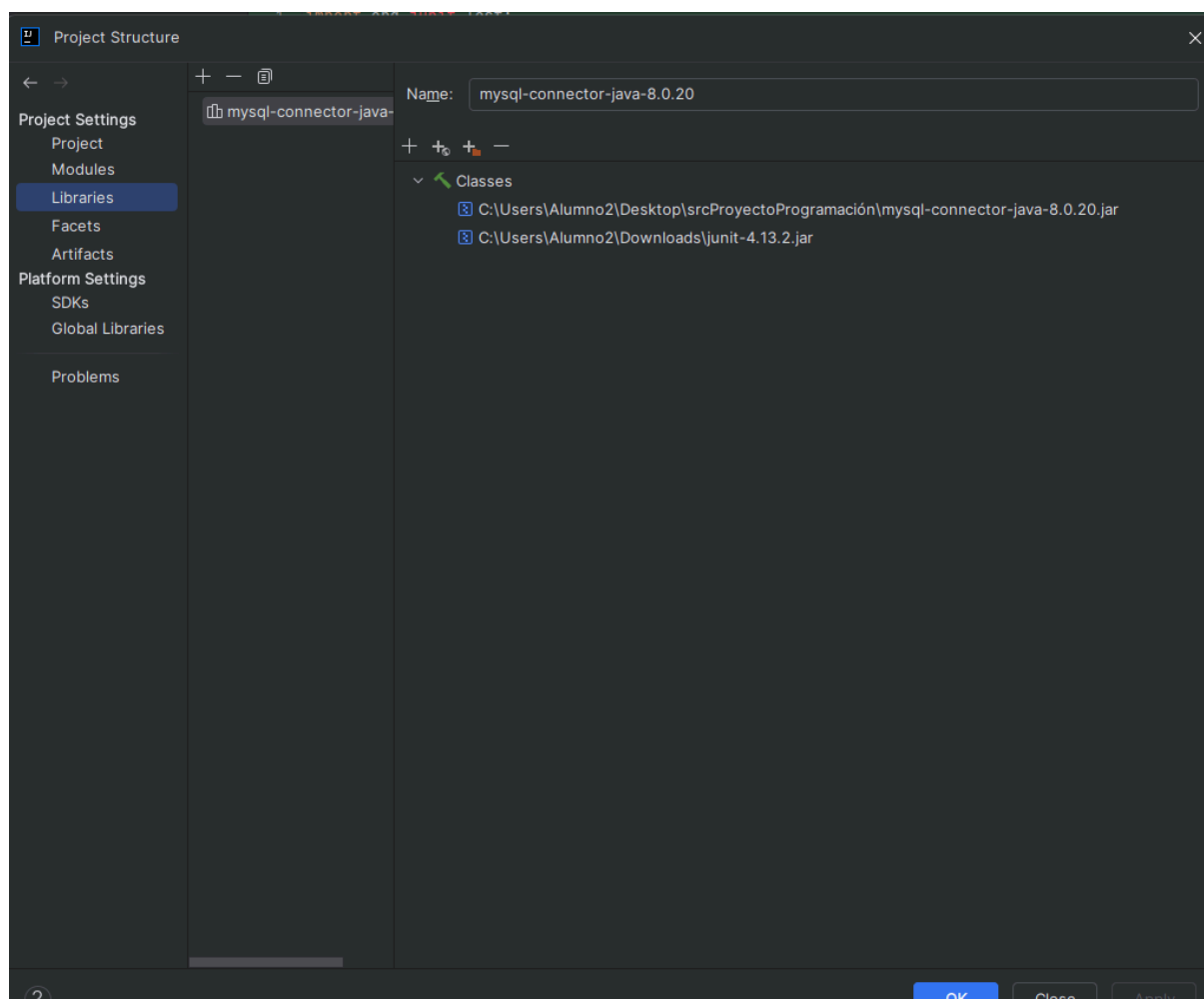
### 2 Añadir JUnit al proyecto

- Copia `junit-4.13.2.jar` en la carpeta `lib` dentro de tu proyecto.
- En IntelliJ, ve a **File > Project Structure > Libraries**.





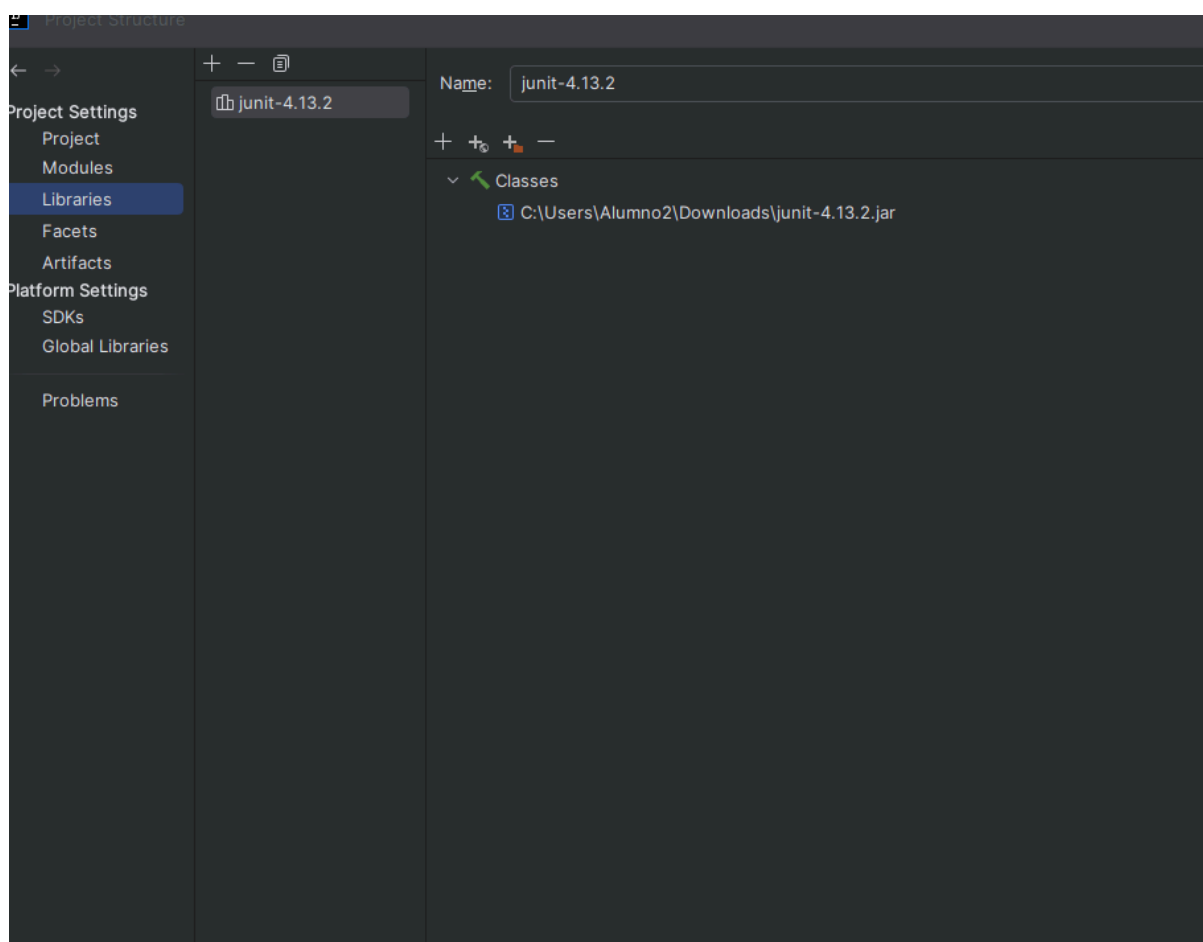
- Presiona **+ Add Library** y selecciona *Add JARs or Directories*.
- Busca el archivo `junit-4.13.2.jar` dentro de `lib`, selecciona **Apply and Close**.



## Crear una clase de prueba en IntelliJ

**Abre IntelliJ IDEA.** Ve a **File > New > Project**. Selecciona **Java** y presiona **Next**. Escribe un nombre para el proyecto (Ejemplo: PruebasJUnit). Presiona **Finish**.

- ♦ **Agregar JUnit al nuevo proyecto**



Ve a **File > Project Structure > Libraries**. Haz clic en **+ Add Library > Add JARs or Directories**. Busca y selecciona `junit-4.13.2.jar` (si no lo tienes, descárgalo aquí). Presiona **Apply and Close**.

**Crear una clase de prueba fácil (Calculadora.java)**



```
n.java  Calculadora.java
public class Calculadora {
    public int sumar(int a, int b) {
        return a + b;
    }
}
```

**Crear la prueba unitaria (CalculadoraTest.java)**



```
import static org.junit.Assert.*;
import org.junit.Test;

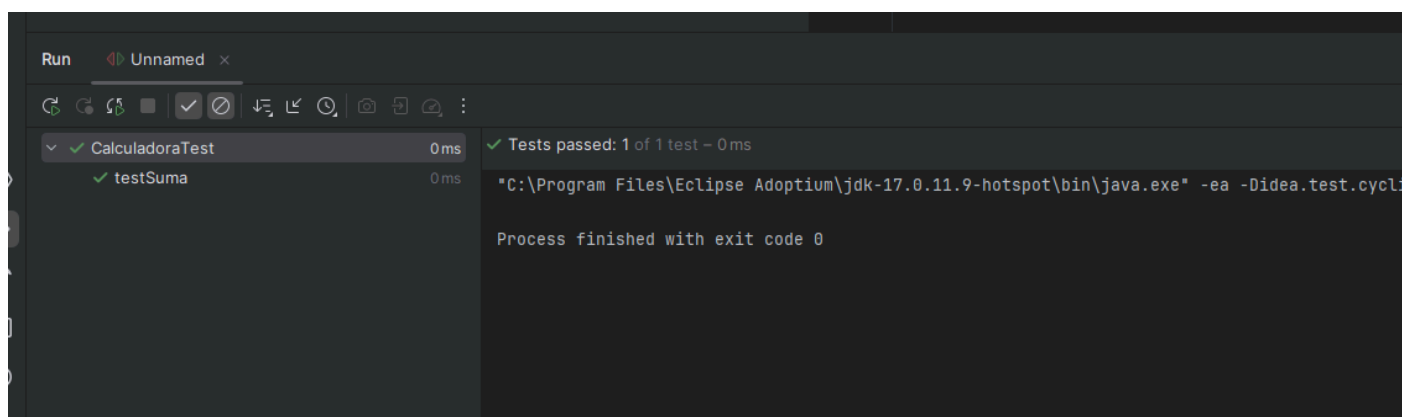
public class CalculadoraTest {

    @Test
    public void testSuma() {
        Calculadora calc = new Calculadora();
        int resultado = calc.sumar(2, 3);
        assertEquals(5, resultado);
    }
}
```

## Ejecutar la prueba en IntelliJ

Haz clic derecho en `CalculadoraTest.java`. Selecciona "Run 'CalculadoraTest'". Si todo está bien, verás verde indicando que la prueba pasó.

```
1 > import ...
3
4 public class CalculadoraTest {
5
6     @Test
7     public void testSuma() {
8         Calculadora calc = new Calculadora();
9         int resultado = calc.sumar(a: 2, b: 3);
10        assertEquals(expected: 5, resultado);
11    }
12 }
13
14
```



**Prueba 1: Validar que la etiqueta <img> tenga el atributo alt correctamente**



```
1  import static org.junit.Assert.*;
2  import org.junit.Test;
3
4  public class ImagenTest {
5      @Test
6      public void testImagenTieneAlt() {
7          String imagenHTML = "<img src='logo.png' alt='Logo de la empresa'>";
8          assertTrue(imagenHTML.contains("alt=")); // Verifica que la imagen tiene el atributo alt
9      }
10 }
11
```

```

:
ns ✓ Tests passed: 1 of 1 test - 0 ms
ns "C:\Program Files\Eclipse Adoptium\jdk-17.0.11.9-hotspot\bin\java.exe" -ea -Didea.test.cyclic.buffer.size=1048576 "-
Process finished with exit code 0
```

## Prueba 2: Verificar que la etiqueta <meta> contenga el atributo charset="UTF-8"

```
© Main.java    © Calculadora.java    © CalculadoraTest.java    © ImagenTest.java    ©
1      import static org.junit.Assert.*;
2      import org.junit.Test;
3
4  >> public class MetaCharsetTest {
5      @Test
6  > public void testMetaCharset() {
7          String metaTag = "<meta charset='UTF-8'>";
8          assertTrue(metaTag.contains("charset='UTF-8'"));
9      }
10 }
11
```

## Depuración y Refactorización

Durante el desarrollo del proyecto, se han utilizado las herramientas de depuración integradas en Eclipse e IntelliJ para localizar y solucionar errores, especialmente en la conexión con la base de datos y la funcionalidad de inserción/búsqueda de libros.

Se han aplicado técnicas de refactorización para mejorar el código:

- **Extracción de métodos** para dividir funciones largas en partes más legibles.
- **Renombrado de variables y métodos** para mejorar la claridad del código (btnGuardar, txtTitulo, insertarLibro(...)).
- **Eliminación de código duplicado** y limpieza de imports innecesarios.



- Mejora de la estructura de clases DAO para que cada clase tenga una única responsabilidad.

## Calidad de Código

**Durante el desarrollo de este proyecto, se ha aplicado una codificación limpia y estructurada siguiendo buenas prácticas de Java:**

- Nombres significativos: Se han utilizado nombres descriptivos para clases como LibroDAO, Ventana\_Insertar, ConexionMySQL, lo que facilita entender su propósito.
- Estructura organizada:
  - Interfaz gráfica con Swing: JFrame, JPanel, JLabel, JTextField, JButton, JCheckBox.
  - Acceso a base de datos mediante clases DAO.
  - Separación clara entre interfaz, lógica y acceso a datos.
- Estilo Java respetado:
  - Uso de camelCase para métodos y variables.
  - Clases con mayúscula inicial.





- Indentación consistente.
- Código legible:
  - Comentarios explicativos donde es necesario.
  - Métodos cortos, que realizan una sola función.
  - Eliminación de código innecesario o sin uso.