



# Tarea de evaluación - Entorno de Desarrollo

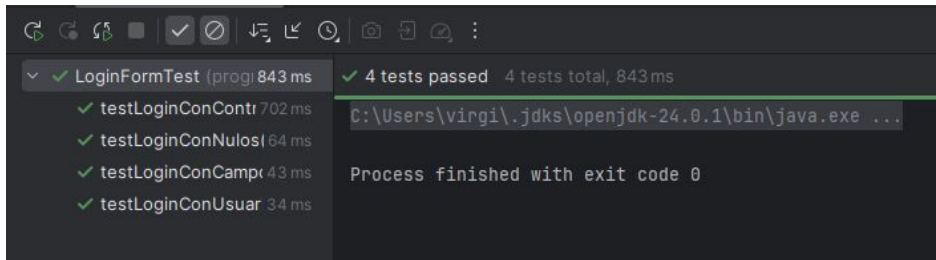
Hecho por:

- Virgilio Jesus Dominguez Gonzalez
- Jose Antonio Jimenez Bernaza
- Sergio Ponce Castro

# Pruebas unitarias

Hemos elegido hacer las pruebas en uno de las clases más importantes de nuestro programa (LoginForm), ya que sin su funcionamiento el programa no cobra sentido.

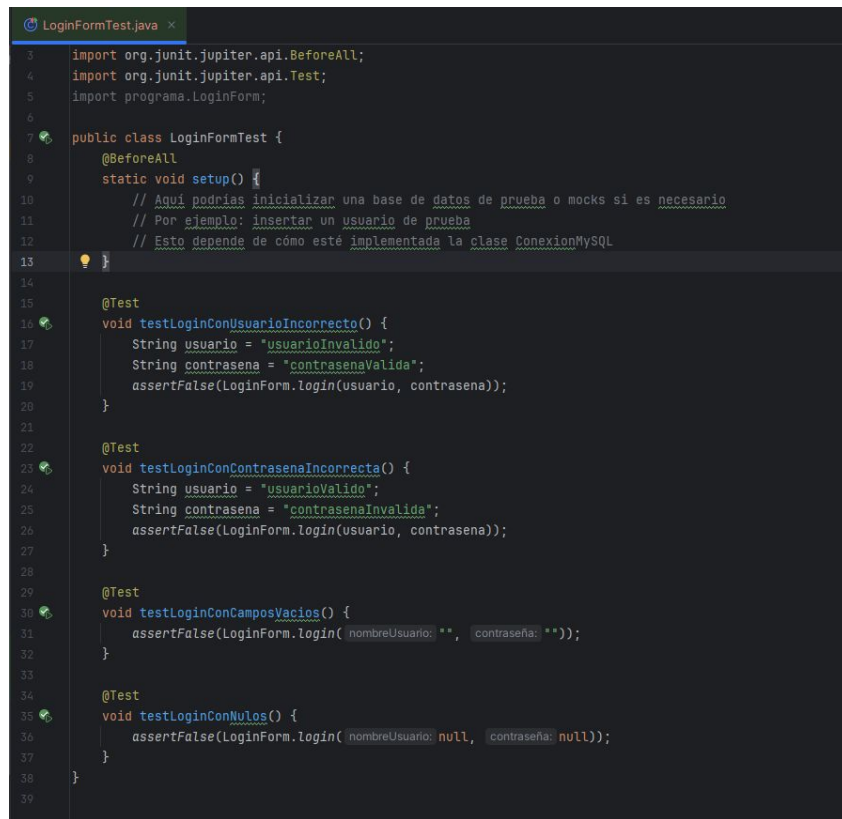
Hemos realizado en total 4 pruebas unitarias, realizadas en IntelliJ, estas se basan en los posibles errores al iniciar sesión de un usuario.



The screenshot shows the IntelliJ test results window for the class LoginFormTest. It indicates that all 4 tests passed successfully. The tests listed are testLoginConContraseñaIncorrecta (702 ms), testLoginConNulos (64 ms), testLoginConCamposVacios (43 ms), and testLoginConUsuarioValido (34 ms). The total execution time for the test class is 843 ms. The process finished with exit code 0.

```
✓ LoginFormTest (progi 843 ms)
  ✓ testLoginConContraseñaIncorrecta 702 ms
  ✓ testLoginConNulos 64 ms
  ✓ testLoginConCamposVacios 43 ms
  ✓ testLoginConUsuarioValido 34 ms

✓ 4 tests passed 4 tests total, 843 ms
C:\Users\virgi\.jdk\openjdk-24.0.1\bin\java.exe ...
Process finished with exit code 0
```



The screenshot shows the source code of the LoginFormTest.java file. It includes imports for JUnit and the LoginForm class. The test class contains four test methods: testLoginConUsuarioIncorrecto, testLoginConContraseñaIncorrecta, testLoginConCamposVacios, and testLoginConNulos. Each test method calls the LoginForm.login method with specific inputs and asserts that the result is false.

```
LoginFormTest.java
3 import org.junit.jupiter.api.BeforeAll;
4 import org.junit.jupiter.api.Test;
5 import programa.LoginForm;
6
7 public class LoginFormTest {
8     @BeforeAll
9     static void setup() {
10         // Aquí podrias inicializar una base de datos de prueba o mocks si es necesario
11         // Por ejemplo: insertar un usuario de prueba
12         // Esto depende de cómo esté implementada la clase ConexionMySQL
13
14
15     @Test
16     void testLoginConUsuarioIncorrecto() {
17         String usuario = "usuarioInvalido";
18         String contraseña = "contraseñaValida";
19         assertFalse(LoginForm.login(usuario, contraseña));
20     }
21
22     @Test
23     void testLoginConContraseñaIncorrecta() {
24         String usuario = "usuarioValido";
25         String contraseña = "contraseñaInvalida";
26         assertFalse(LoginForm.login(usuario, contraseña));
27     }
28
29     @Test
30     void testLoginConCamposVacios() {
31         assertFalse(LoginForm.login( "", contraseña: ""));
32     }
33
34     @Test
35     void testLoginConNulos() {
36         assertFalse(LoginForm.login( nombreUsuario: null, contraseña: null));
37     }
38 }
39
```

# Depuración y Refactorización

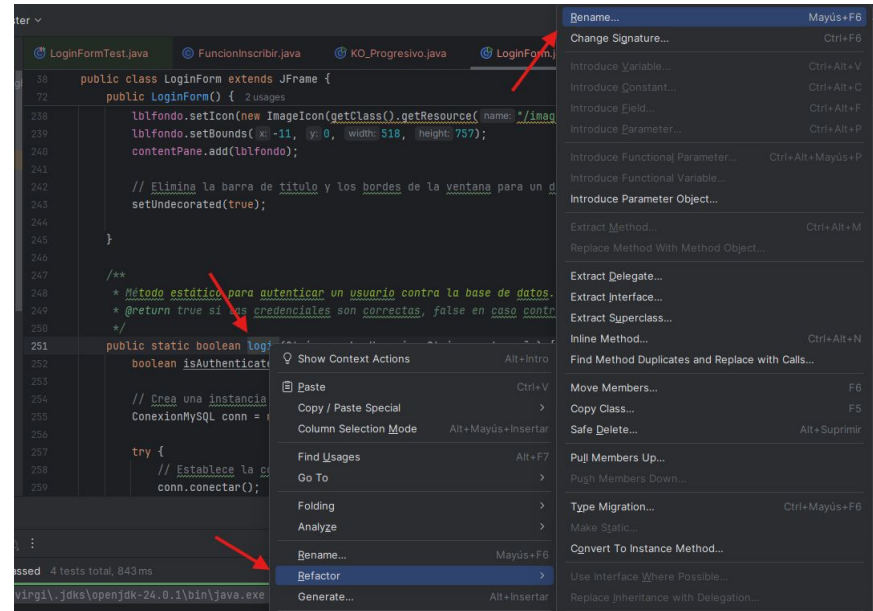
En nuestro caso no hemos necesitado el uso de Refactorización del código ya que hemos llevado siempre una buena estructura en el proyecto y mediante git hemos podido tener buena comunicación y desarrollo por partes.

En este caso vamos a enseñar un caso de como habríamos hecho una Refactorización en el código.

Usaremos de ejemplo el método login().

```
public static boolean login(String nombreUsuario, String contraseña) { 5 usages
    boolean isAuthenticated = false;
```

Necesitaremos hacer clic derecho sobre nuestro método “login()” y elegir la opción “Refactor”, en nuestro caso solo vamos a hacer uso de la característica “Rename” y simplemente escribiremos el nuevo nombre que le daremos a ese método.

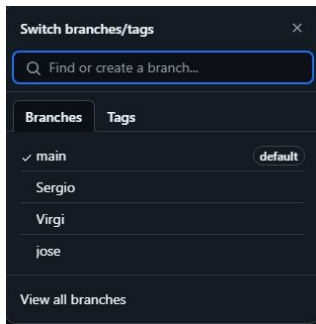


# Git y Github

Nuestro proyecto ha sido sostenido mediante github a la hora de hacer cambios en el código cada uno por una parte diferente.

Para ello hemos realizado un repositorio de nuestro programa y dividido por asignaturas, luego se han creado 3 ramas a parte de la "main" denominadas "Virgi", "Sergio" y "jose".

Gracias a "Git" hemos ido usando comandos básicos de consola "Git Bash" con la que ir realizando "add", "commit" y "push".



```
virgi@Equipo MINGW64 ~/Desktop
$ git clone https://github.com/Virgijdg334/TightPoker/tree/Virgi
fatal: repository 'https://github.com/Virgijdg334/TightPoker/tree/Virgi' does not exist

virgi@Equipo MINGW64 ~/Desktop
$ git clone https://github.com/Virgijdg334/TightPoker
Cloning into 'TightPoker'...
remote: Enumerating objects: 784, done.
remote: Counting objects: 100% (218/218), done.
remote: Compressing objects: 100% (215/215), done.
remote: Total 784 (delta 124), reused 0 (delta 0), pack-reused 566 (from 1)
Receiving objects: 100% (784/784), 3.77 MiB | 13.25 MiB/s, done.
Resolving deltas: 100% (427/427), done.

virgi@Equipo MINGW64 ~/Desktop
$ cd TightPoker/

virgi@Equipo MINGW64 ~/Desktop/TightPoker (main)
$ git checkout Virgi
branch 'Virgi' set up to track 'origin/Virgi'.
Switched to a new branch 'Virgi'

virgi@Equipo MINGW64 ~/Desktop/TightPoker (Virgi)
$ git add .

virgi@Equipo MINGW64 ~/Desktop/TightPoker (Virgi)
$ git commit -m "Actualizacion"
On branch Virgi
Your branch is up to date with 'origin/Virgi'.

nothing to commit, working tree clean

virgi@Equipo MINGW64 ~/Desktop/TightPoker (Virgi)
$ git status
On branch Virgi
Your branch is up to date with 'origin/Virgi'.

nothing to commit, working tree clean

virgi@Equipo MINGW64 ~/Desktop/TightPoker (Virgi)
$ git push origin Virgi
Everything up-to-date
```

