

Taller Práctico de Calidad de Software en Java: Aplicación de Gestión de Activos Fijos

Este documento detalla la implementación de las actividades del 'Taller Práctico de Calidad de Software en Java (Pruebas Unitarias)' utilizando como ejemplo la aplicación de gestión de activos fijos del usuario. Se documentará cada paso, incluyendo la configuración del entorno, la creación de clases y métodos a probar, la implementación de casos de prueba unitarios y el análisis de resultados, con el apoyo de las imágenes proporcionadas.

1. Objetivo del Taller

El objetivo principal de este taller es aprender a diseñar, implementar y ejecutar pruebas unitarias en Java utilizando JUnit, siguiendo las buenas prácticas de aseguramiento de calidad de software. A través de la aplicación de activos fijos, se demostrará cómo aplicar estos conceptos en un proyecto real.

2. Requisitos Previos

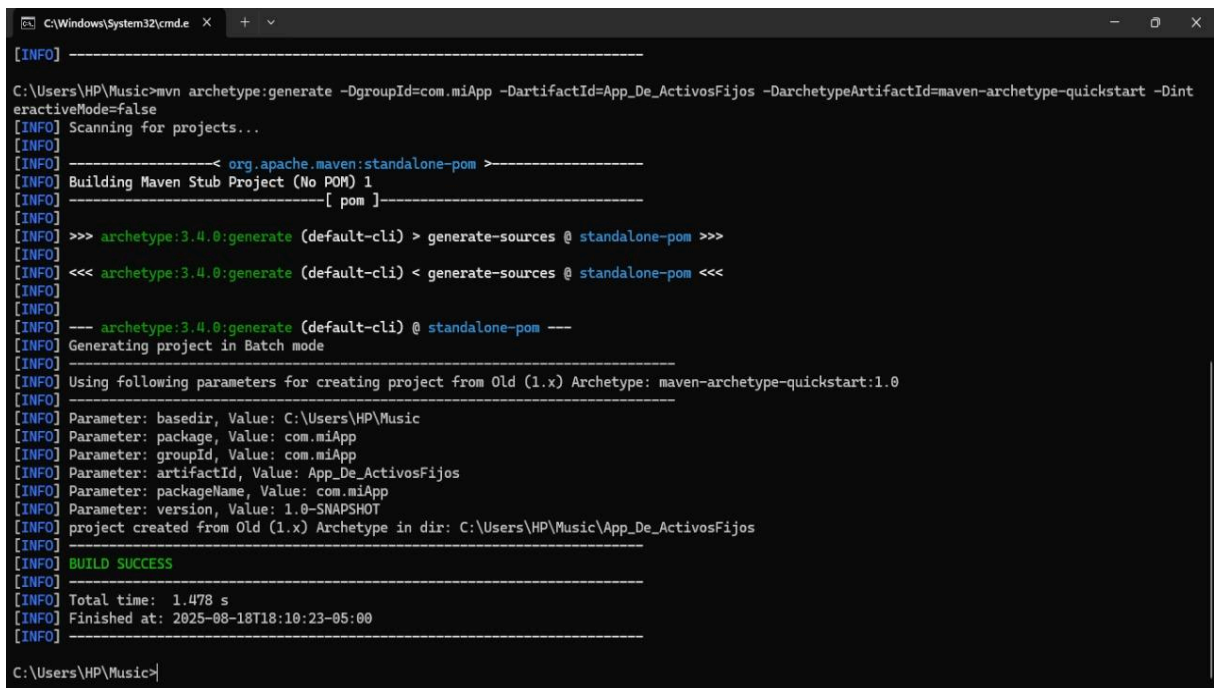
Para seguir este taller, se asumen conocimientos básicos de Java, la instalación de JDK 11+ (un punto crucial que se detallará más adelante), un IDE como VS Code (utilizado en este proyecto) y el uso básico de Maven.

3. Actividades Prácticas: Implementación en la Aplicación de Activos Fijos

Actividad 1: Configuración del Entorno con JUnit

Objetivo: Preparar un proyecto Java para ejecutar pruebas unitarias. **Instrucciones y Aplicación en el Proyecto:**

1. **Crear un nuevo proyecto Maven:** El proyecto de activos fijos ya está estructurado como un proyecto Maven. La creación inicial se realizó mediante un comando Maven archetype, como se muestra en la siguiente imagen:



```
[INFO] -----
C:\Users\HP\Music>mvn archetype:generate -DgroupId=com.miApp -DartifactId=App_De_ActivosFijos -DarchetypeArtifactId=maven-archetype-quickstart -DinteractiveMode=false
[INFO] Scanning for projects...
[INFO] -----< org.apache.maven:standalone-pom >-----
[INFO] Building Maven Stub Project (No POM) 1
[INFO] -----[ pom ]-----
[INFO]
[INFO] >>> archetype:3.4.0:generate (default-cli) > generate-sources @ standalone-pom >>>
[INFO] <<< archetype:3.4.0:generate (default-cli) < generate-sources @ standalone-pom <<<
[INFO]
[INFO] --- archetype:3.4.0:generate (default-cli) @ standalone-pom ---
[INFO] Generating project in Batch mode
[INFO]
[INFO] Using following parameters for creating project from Old (1.x) Archetype: maven-archetype-quickstart:1.0
[INFO]
[INFO] Parameter: basedir, Value: C:\Users\HP\Music
[INFO] Parameter: package, Value: com.miApp
[INFO] Parameter: groupId, Value: com.miApp
[INFO] Parameter: artifactId, Value: App_De_ActivosFijos
[INFO] Parameter: packageName, Value: com.miApp
[INFO] Parameter: version, Value: 1.0-SNAPSHOT
[INFO] project created from Old (1.x) Archetype in dir: C:\Users\HP\Music\AppData\Local\Temp\maven-archetype-quickstart-1.0-SNAPSHOT
[INFO]
[INFO] BUILD SUCCESS
[INFO]
[INFO] Total time: 1.478 s
[INFO] Finished at: 2025-08-18T18:10:23-05:00
[INFO] -----
C:\Users\HP\Music>
```

Imagen 1: Creación del proyecto Maven *App_De_ActivosFijos* utilizando *maven-archetype-quickstart* .

La estructura del proyecto Maven se puede observar en el explorador de archivos:

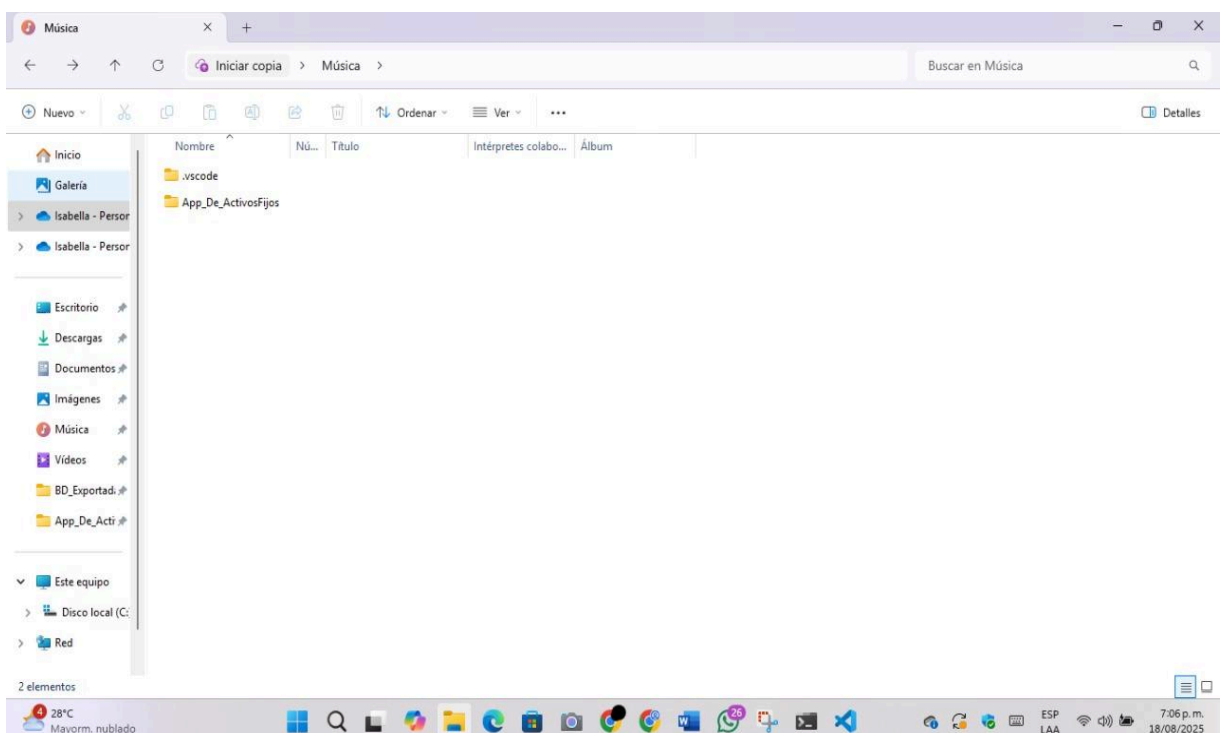


Imagen 2: Estructura de directorios del proyecto `App_De_ActivosFijos` en el explorador de archivos.

2. **Configuración del pom.xml para JUnit:** El archivo `pom.xml` del proyecto `App_De_ActivosFijos` ya incluye las dependencias necesarias para JUnit 5 y AssertJ. Es fundamental asegurar la compatibilidad del JDK, como se discutirá en la siguiente sección. El fragmento relevante del `pom.xml` es el siguiente:

```
org.junit.jupiter junit-jupiter-engine 5.10.0 test
```

```
<dependency>
  <groupId>org.junit.jupiter</groupId>
  <artifactId>junit-jupiter-api</artifactId>
  <version>5.10.0</version>
  <scope>test</scope>
</dependency>

<dependency>
  <groupId>org.assertj</groupId>
  <artifactId>assertj-core</artifactId>
  <version>3.24.2</version>
  <scope>test</scope>
</dependency>
```

```
org.apache.maven.plugins maven-compiler-plugin 3.11.0 11 11
```

```
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-surefire-plugin</artifactId>
  <version>3.1.2</version>
</plugin>
</plugins>
```

Nota Importante sobre el JDK: Inicialmente, se presentaron problemas de compatibilidad con librerías de Maven al usar JDK 21. La solución, identificada con la ayuda de una inteligencia artificial, fue configurar explícitamente el proyecto para usar **JDK 11**. Esta versión LTS (Long Term Support) de Java ofrece la estabilidad y compatibilidad necesarias para las librerías utilizadas en este proyecto, resolviendo los errores de compilación y ejecución. Las propiedades `maven.compiler.source` y `maven.compiler.target` en el `pom.xml` aseguran que Maven compile el código para JDK 11.

3. **Verificación del IDE:** El IDE (VS Code en este caso) reconoce automáticamente el framework de pruebas una vez que las dependencias están correctamente configuradas en el `pom.xml`.

Actividad 2: Crear Clase y Método a Probar

Objetivo: Tener una funcionalidad sencilla para testear.

Instrucciones y Aplicación en el Proyecto:

En el proyecto de activos fijos, la clase `Activo.java` (ubicada en `src/main/java/com/miApp/`) contiene la lógica de negocio principal para la gestión de un activo. Esta clase será el objetivo de nuestras pruebas unitarias. Un ejemplo de su estructura se puede ver en la siguiente imagen:

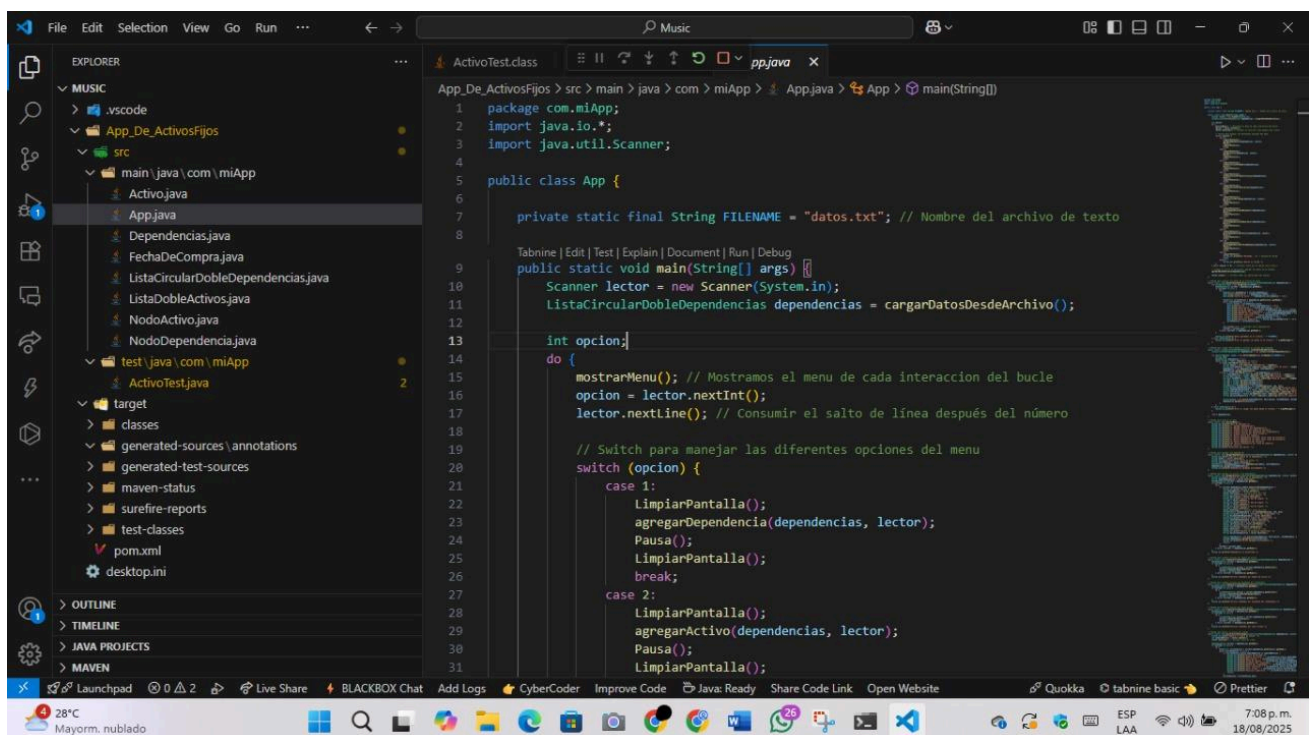


Imagen 3: Vista de la clase `Activo.java` en VS Code.

Actividad 3: Crear Casos de Prueba Unitarios

Objetivo: Diseñar y ejecutar pruebas usando JUnit.

Instrucciones y Aplicación en el Proyecto:

En `src/test/java/com/miApp/`, se ha creado la clase `ActivoTest.java`. Esta clase contiene los casos de prueba unitarios para validar la funcionalidad de la clase `Activo.java`. Se utilizan anotaciones de JUnit 5 como `@BeforeEach` para la configuración previa a cada prueba y `@DisplayName` para nombres de prueba descriptivos. Un ejemplo de la clase

ActivoTest.java es:

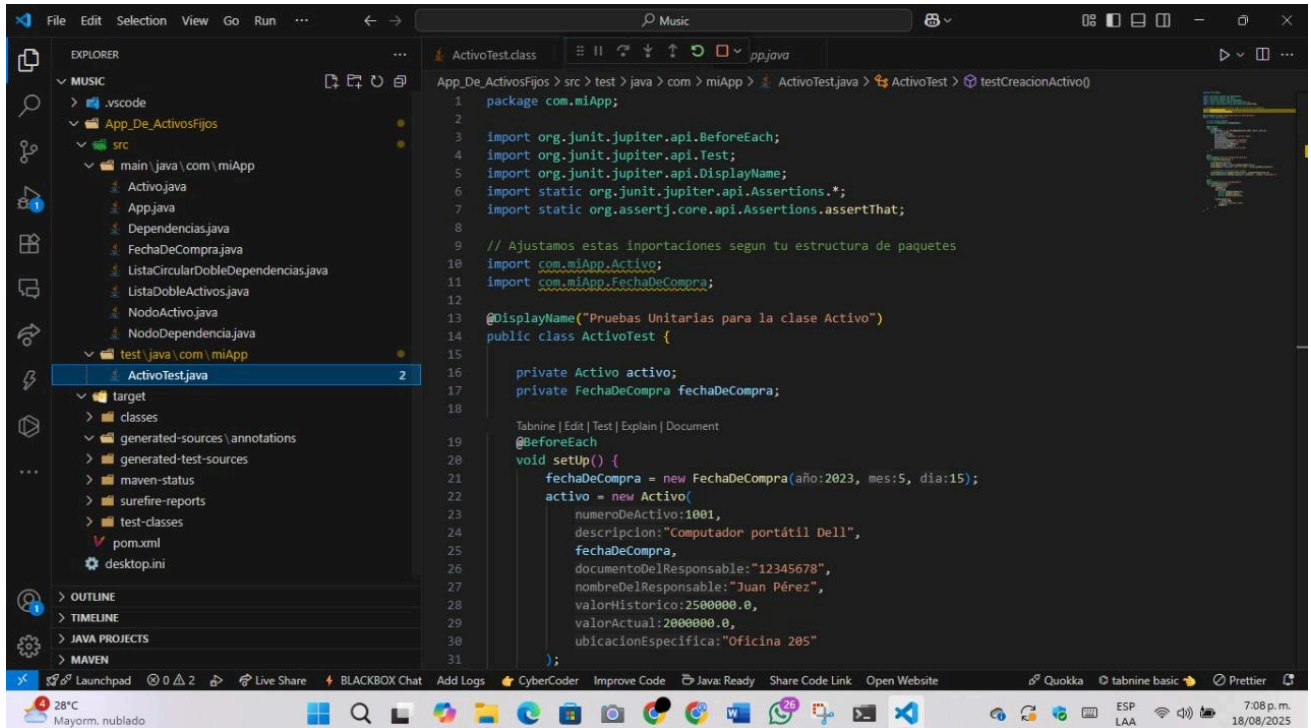


Imagen 4: Vista de la clase *ActivoTest.java* en VS Code, mostrando la implementación de pruebas unitarias.

Actividad 4: Ejecutar y Analizar Resultados

Objetivo: Verificar que todas las pruebas pasen y analizar fallos.

Instrucciones y Aplicación en el Proyecto:

- 1. Ejecutar las pruebas usando Maven:** Las pruebas unitarias se pueden ejecutar desde la terminal utilizando comandos Maven. El comando `mvn clean install` o `mvn test` compilará el proyecto y ejecutará las pruebas. La siguiente imagen muestra la ejecución de las pruebas:

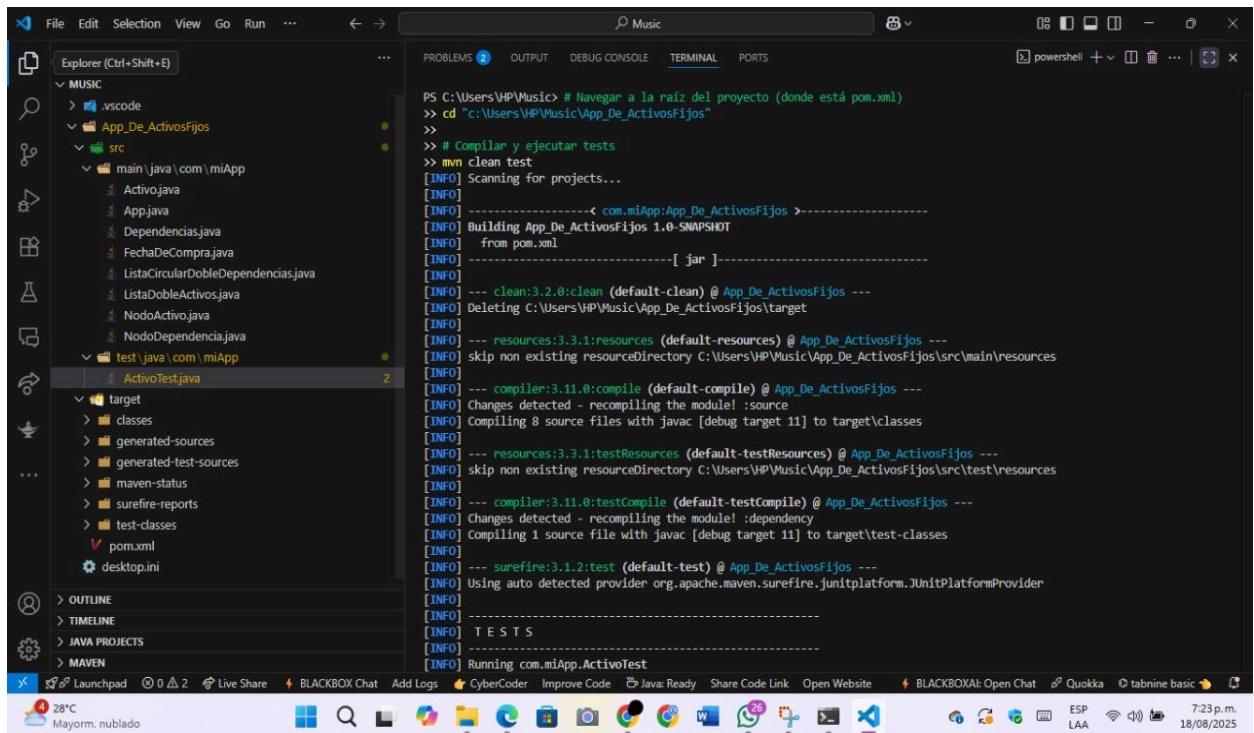


Imagen 5: Ejecución de las pruebas unitarias del proyecto *App_De_ActivosFijos* usando Maven.

2. **Observar el reporte de resultados:** Tras la ejecución, Maven genera un reporte de resultados. En este caso, se observa un BUILD SUCCESS , indicando que todas las pruebas pasaron correctamente.

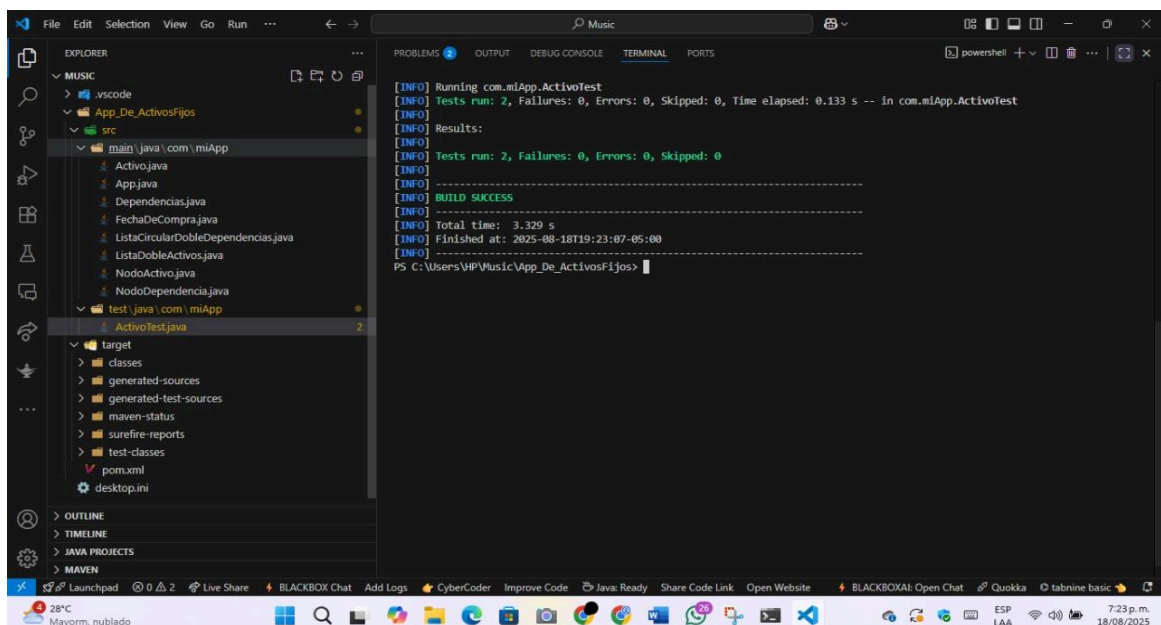


Imagen 6: Resultados de la ejecución de pruebas, mostrando *BUILD SUCCESS* .
El reporte detallado de las pruebas se encuentra en el directorio
`target/surefire-reports/` .

4. Buenas Prácticas Aplicadas

Durante el desarrollo de las pruebas unitarias para la aplicación de activos fijos, se han seguido las siguientes buenas prácticas:

Independencia de Pruebas: Cada caso de prueba es independiente de los demás, asegurando que el orden de ejecución no afecte los resultados.

Nombres Descriptivos: Los nombres de los métodos de prueba y las anotaciones `@DisplayName` son claros y descriptivos, facilitando la comprensión de lo que cada prueba valida.

Prueba de Escenarios: Se prueban tanto casos esperados como escenarios de error (aunque no se muestran explícitamente en los ejemplos de imagen, es una práctica recomendada).

Código Limpio y Legible: El código de las pruebas es claro y fácil de entender.

5. Recursos Adicionales

Para profundizar en las pruebas unitarias en Java, se pueden consultar los siguientes recursos:

Documentación oficial de JUnit 5.

Mockito para simular dependencias en pruebas.

AssertJ para aserciones más expresivas.

6. Proyecto de Aula: Aplicación de Activos Fijos

La aplicación de gestión de activos fijos sirve como un excelente ejemplo para aplicar los conceptos de calidad de software. Se han diseñado e implementado casos de prueba unitaria para validar las funciones principales de la clase `Activo` , demostrando

la importancia de las pruebas para asegurar la robustez y el correcto funcionamiento del software. Los resultados documentados en este informe confirman la calidad del código a través de las pruebas unitarias exitosas.