



Programa de JUnit
JUnit



Academia Java
Entregable

Natalia Esquivel Ocadiz
19 de septiembre 2025

Introducción:

En este proyecto se realizaron pruebas unitarias con JUnit Jupiter 5 al proyecto de Inyección de Dependencias (IoD), para comprobar que los métodos del código funcionaran correctamente.

Objetivos:

- Comprobar que el código funciona correctamente y/o detectar errores tempranamente y modificar el código para corregirlos
- Implementar pruebas de JUnit para probar partes específicas del proyecto IoD
- Cumplir con un coverage de al menos 80% del código para asegurar una buena calidad del mismo

Implementación: Explicación concisa de la solución

1. Se crea un proyecto Maven
2. Se revisa que el pom.xml contenga las dependencias de JUnit-Jupiter, y para asegurar que esté cargada correctamente la dependencia se procede a actualizar el proyecto
3. En el package de test, es donde se encuentra la clase test donde se realizan las pruebas
4. Se implementan las pruebas unitarias con el siguiente método:



Con el fin de tener una mejor organización, se usaron las anotaciones `@BeforeEach` y `@AfterEach` para evitar repeticiones innecesarias al ser código que todos los métodos tienen en común

- `@BeforeEach`: lo que hace es capturar la salida de la consola
- `@AfterEach`: lo que hace es limpiar la captura de la salida de la consola para la siguiente prueba
- Se instancia de la clase a probar, como Guerrero, Espada, Hechicero, BastonMagico, etc.

Set up:

Se hacen la anotación `@Test` antes de test, para indicar que se trata de un método de prueba.

La anotación `@DisplayName` permite darle un nombre a la prueba, los nombres dados denotan explícitamente qué es lo que hace la prueba.

Se crea la instancia de la clase a probar

Execute

Se mandan a llamar los métodos que se buscan probar, por ejemplo `guerrero.atacar()`.

Se probaron los métodos de ataque y defensa de cada personaje, la clase Batalla Arena que simula la batalla de dos personajes y el método main.


Assert

Los asserts empleado para revisar que los resultados sean los que se esperan son:

- `Assertions.assertTrue(actual.contains(...))` para asegurar que se imprima el resultado esperado

Resultados: Evidencia de que funciona correctamente

- Cobertura de pruebas: 85.7% de coverage

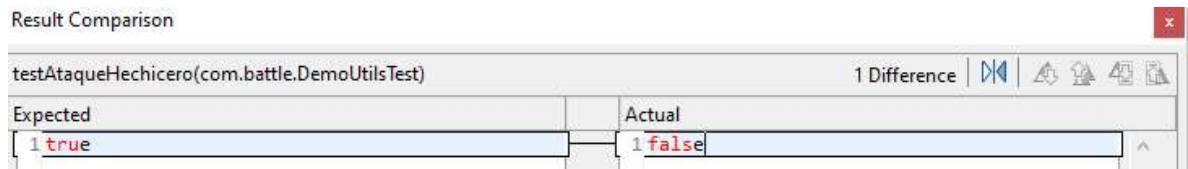
DemoUtilsTest (2) (19 sept 2025 13:24:40)				
Element	Coverage	Covered Instructio...	Missed Instructions	Total Instructions
> junit4	 85.7 %	384	64	448

- Casos de prueba edge cases y escenarios negativos

Se prueba el caso de que Hechicero pueda devolver un “ataque” diferente al esperado. Se espera que la prueba falle.

```
@Test
@DisplayName("Ataque Hechicero")
void testAtaqueHechicero() {
    //set up
    bastonMagico = new BastonMagico();
    hechicero = new Hechicero(bastonMagico);

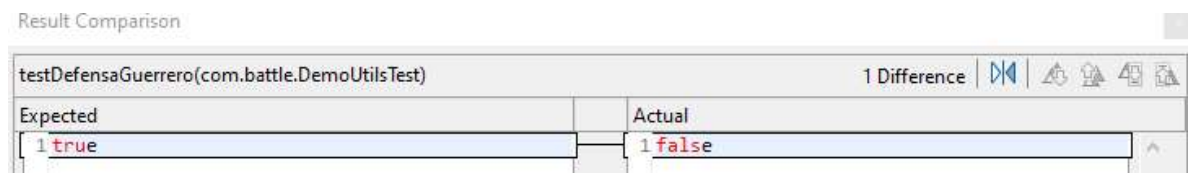
    // execute
    hechicero.atacar();
    String actual1 = outputStream.toString();
    //assert
    Assertions.assertTrue(actual1.contains("El hechicero lanza flechas con un arco"));
} //testAtaqueHechicero
```



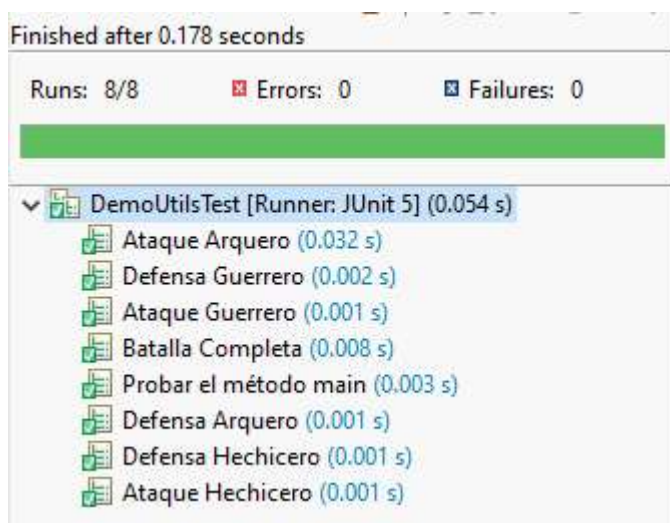
Se prueba el caso de que Guerrero pueda devolver la “defensa” del arquero. Se espera que la prueba falle ya que debería devolver únicamente la defensa del guerrero.

```
@Test
@DisplayName("Defensa Guerrero")
void testDefensaGuerrero() {
    //set up
    espada = new Espada();
    guerrero = new Guerrero(espada);

    // execute
    guerrero.defender();
    String actual = outputStream.toString();
    //assert
    Assertions.assertTrue(actual.contains("El guerrero se defiende con un hechizo de protección"));
} //testDefensaGuerrero
```



- Screenshots de resultados de ejecución: se realizaron 8 pruebas



Reflexiones: Las pruebas unitarias con JUnit 5 han demostrado ser una herramienta fundamental en el desarrollo de software de calidad. Durante la implementación de este proyecto se obtuvieron las siguientes lecciones clave:

Detección temprana de errores: JUnit permitió identificar y corregir el ataque del hechicero, el cual era incorrecto y había pasado desapercibido durante el desarrollo inicial. Esto confirma la importancia de implementar pruebas desde las primeras etapas del proyecto.

Cobertura como métrica de calidad: Se cumplió con el coverage mínimo establecido, proporcionando confianza en la funcionalidad del código. Sin embargo, se identificaron oportunidades de mejora, como implementar pruebas que validen que no se acepte un personaje con valor null y probar la respuesta de personajes con distintas combinaciones de armas para asegurar mayor robustez del sistema.

Diseño de pruebas efectivas: El uso de anotaciones como `@BeforeEach` y `@AfterEach` mejoró significativamente la organización del código de pruebas, eliminando duplicación innecesaria y asegurando un estado limpio para cada test. Esta práctica resultó en un código más legible.

Esta experiencia refuerza la importancia de adoptar las pruebas unitarias como práctica estándar para garantizar la calidad y confiabilidad del software desarrollado.