**Prueba de Software**

TP N° 1: Pruebas Unitarias, Integración y Regresión.



Universidad Nacional de Lanús.

Lic. Sistemas

Ciclo Lectivo 2024.

**Docentes:**

M. Ing. Eduardo Diez

Lic. Pablo San Roman

**Estudiantes:**

– Agustina Tarifa

– Facundo Galli

– Carolina Aguirre

– Agustín Garabito

**Investigación sobre Mockito:**

Para utilizar la librería Mockito se debe instalar el IDE Eclipse, una vez instalado debemos crear un nuevo proyecto utilizando Maven

El pom.xml debería verse algo tal que así:

1. <?xml version="1.0" encoding="UTF-8"?>

2. <project xmlns="http://maven.apache.org/POM/4.0.0"

3. xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

4. xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">

5. <modelVersion>4.0.0</modelVersion>

6. <groupId>org.example</groupId>

7. <artifactId>m3-junit</artifactId>

8. <version>1.0-SNAPSHOT</version>

9. <properties>

10. <maven.compiler.source>11</maven.compiler.source>

11. <maven.compiler.target>11</maven.compiler.target>

12. </properties>

13. <dependencies>

14.

15. <dependency>

16. <groupId>org.junit.jupiter</groupId>

17. <artifactId>junit-jupiter</artifactId>

18. <version>5.8.1</version>

19. <scope>test</scope>

20. </dependency>

21.

22. <dependency>

23. <groupId>org.junit.platform</groupId>

24. <artifactId>junit-platform-runner</artifactId>

25. <version>1.8.1</version>

26. <scope>test</scope>

27. </dependency>

28.

29. <dependency>

30. <groupId>org.mockito</groupId>

31. <artifactId>mockito-junit-jupiter</artifactId>

32. <version>4.0.0</version>

33. <scope>test</scope>

34. </dependency>

35. </dependencies>

36. </project>

37.

**Actualizar el proyecto haciendo:**

Click derecho en el proyecto -> Maven -> Update Project

**Importar la librería en los test:**

1. import static org.mockito.Mockito.\*;

**Conceptos básicos de Mockito**

¿Qué es Mockito?

Mockito es una biblioteca de Java diseñada para facilitar las pruebas unitarias. Permite crear objetos simulados (mock objects) que representan clases, lo que ayuda a verificar el comportamiento de otras partes del código sin depender de implementaciones reales.

Con Mockito, se puede:

1. **Simular Dependencias**: Generar versiones ficticias de objetos que son complicados de usar en las pruebas, como conexiones a bases de datos o servicios externos.
2. **Verificar Interacciones**: Asegurarte de que ciertos métodos se hayan llamado con los parámetros adecuados.
3. **Definir Comportamientos**: Especificar qué debe devolver un método simulado al ser invocado.

Es ampliamente utilizado en el desarrollo de software en Java debido a su facilidad de uso y efectividad.

Creación de mocks:

1. producto1 = mock(Producto.class);

2. producto2 = mock(Producto.class);

Definición de Comportamientos:

1. when(producto1.getIdProducto()).thenReturn(1);

2. when(producto1.getProducto()).thenReturn("Producto 1");

3. when(producto1.getPrecio()).thenReturn(20.0f);

4. // VALORES PARA DEVOLVER

1. when(productoMock.getNombre()).thenThrow(new RuntimeException("Error"));

2. // DEVUELVE EXCEPCIÓN

Verificaciones:

1. verify(productoMock).getPrecio();

2.  // VERIFICA SI FUE LLAMADO EL MÉTODO

1. verify(productoMock, times(2)).getId();

2.  // CUANTAS VECES FUE LLAMADO EL MÉTODO

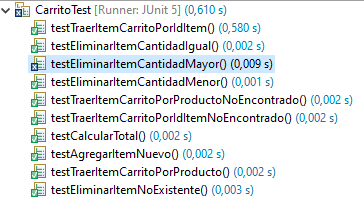
Reinicio:

1. reset(productoMock);

2. // REINICIA EL ESTADO DE UN MOCK

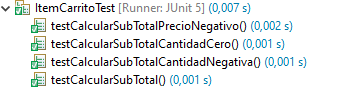
**Script de las pruebas:**

CarritoTest:



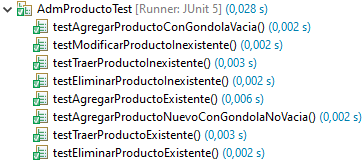
Se detectó un error al intentar eliminar más stock del que ya hay

ItemCarritoTest:



Todos los test pasaron correctamente

AdmProductoTest:



Todos los test pasaron correctamente

**Cobertura de código:**

**Link al repositorio de GitHub:**

<https://github.com/AgustinGarabito/PruebaSoftware-G4>