

Framework Mockito

Mockito

Mockito es un framework de simulación (mocking) en Java ampliamente utilizado en el desarrollo de pruebas unitarias. Proporciona una forma sencilla y flexible de crear objetos simulados, especificar su comportamiento y verificar las interacciones con ellos.

Instalar mockito

Ahora que usamos Maven solo necesitamos agregar la dependencia de mockito al pom.xml y él se encargará de descargar la librería para usarlo en nuestro proyecto.

```
...  
<dependency>  
  <groupId>org.mockito</groupId>  
  <artifactId>mockito-core</artifactId>  
  <version>5.4.0</version>  
  <scope>test</scope>  
</dependency>  
...
```

Cómo se usa Mockito

- **Creación de objetos simulados (mocks):** En Mockito, puedes crear objetos simulados utilizando el método `mock()`. Estos objetos simulados actúan como sustitutos de los objetos reales y pueden ser interfaces, clases concretas o abstractas (las interfaces y clases abstractas las veremos en las clases siguientes).
- **Configuración de comportamientos:** Una vez que tienes un objeto simulado, puedes especificar su comportamiento utilizando métodos

como `when().thenReturn()` o `doAnswer()`. Esto te permite definir qué debe hacer el objeto simulado cuando se llame a un método específico.

Creación de objetos simulados y configuración de comportamientos:

```
@Test
public void testMockitoMockCreationAndBehaviour() {
    // Creación del mock
    List<String> mockedList = mock(List.class);

    // Configuración del comportamiento
    when(mockedList.get(0)).thenReturn("primer elemento");

    // Prueba
    String firstElement = mockedList.get(0);
    assertEquals("primer elemento", firstElement);

    // Cuando se llama a un método que no ha sido configurado, Mockito
    // devuelve null para los objetos
    // y los valores predeterminados para los tipos primitivos
    String secondElement = mockedList.get(1);
    assertNull(secondElement);
}
```

- **Verificación de interacciones:** Con Mockito, puedes verificar si se han realizado ciertas interacciones con los objetos simulados. Puedes comprobar si un método se ha llamado, cuántas veces se ha llamado y con qué argumentos se ha llamado utilizando el **método `verify()`**.

```
@Test
public void testMockitoInteractionVerification() {
    // Creación del mock
    List<String> mockedList = mock(List.class);

    // Interacción con el mock
    mockedList.add("un elemento");

    // Verificación de la interacción
    verify(mockedList).add("un elemento");
}
```

- **Argumentos de captura:** Mockito también permite capturar los argumentos pasados a los métodos llamados en los objetos simulados. Esto es útil cuando necesitas verificar el valor de los argumentos pasados o hacer aserciones más complejas.

```
@Test
public void testMockitoArgumentCaptor() {
    // Creación del mock
    List<String> mockedList = mock(List.class);

    // Interacción con el mock
    mockedList.add("un elemento");

    // Captura de argumentos
    ArgumentCaptor<String> argCaptor =
ArgumentCaptor.forClass(String.class);
    verify(mockedList).add(argCaptor.capture());

    // Prueba
    assertEquals("un elemento", argCaptor.getValue());
}
```

- **Creación de objetos espías:** En Mockito, un espía (spy) es una característica que permite rastrear y, opcionalmente, modificar el comportamiento de un objeto real durante las pruebas. A diferencia de los objetos simulados (mocks), los espías conservan la implementación original del objeto y solo reemplazan o registran ciertos métodos según sea necesario. En los mocks si no reemplazamos un método y lo invocamos, devolverá el valor por defecto del tipo del dato que devuelve el método, 0 si devuelve un int por ejemplo.

```
@Test
public void testMockitoSpy() {
    // Creación del espía
    List<String> spyList = spy(new ArrayList<String>());

    // Interacción con el espía
    spyList.add("elemento real");

    // Verificación de la interacción
```

```

verify(spyList).add("elemento real");

// Prueba
assertEquals(1, spyList.size());
assertEquals("elemento real", spyList.get(0));
}

```

- **Simulación de dependencias:** Uno de los beneficios clave de Mockito es la simulación de dependencias externas. Puedes simular objetos y servicios externos para aislar la unidad que estás probando y centrarte en su comportamiento específico. Esto se logra reemplazando las dependencias reales con objetos simulados en las pruebas.

```

public class OrderProcessor {
    private PaymentGateway paymentGateway;

    public OrderProcessor(PaymentGateway paymentGateway) {
        this.paymentGateway = paymentGateway;
    }

    public boolean processOrder(double amount) {
        return paymentGateway.processPayment(amount);
    }
}

public class OrderProcessorTest {

    @Test
    public void testOrderProcessor() {
        // Crear un objeto simulado de PaymentGateway
        PaymentGateway mockPaymentGateway =
mock(PaymentGateway.class);

        // Configurar el comportamiento del objeto simulado

when(mockPaymentGateway.processPayment(100.0)).thenReturn(true);

        // Inyectar la dependencia simulada en OrderProcessor
        OrderProcessor orderProcessor = new
OrderProcessor(mockPaymentGateway);

        // Probar la funcionalidad de OrderProcessor
        boolean result = orderProcessor.processOrder(100.0);

```

```
// Verificar la interacción y resultado
verify(mockPaymentGateway).processPayment(100.0);
assertTrue(result);
}
}
```

Aquí, estamos simulando la clase `PaymentGateway` y la inyectamos en `OrderProcessor`. La prueba verifica que `OrderProcessor` interactúa correctamente con `PaymentGateway` y devuelve el resultado esperado.

Esta inyección de dependencia ayuda a probar la clase `OrderProcessor` en aislamiento, sin la necesidad de una implementación real de `PaymentGateway`. Esto hace que las pruebas sean más rápidas y confiables, y permite enfocarse en la lógica y el comportamiento específicos de la clase que estás probando.

💡 La **inyección de dependencias** se utiliza generalmente cuando se implementa el patrón experto. En caso de que quieras profundizar sobre este tema te dejamos acceso a este documento: 👉 [patrón experto](#)