TESTS CON MOCKITO

Mockito es un framework de pruebas unitarias que permite crear objetos simulados (mocks) con una API limpia y simple haciendo que las pruebas sean legibles.

Cuando tenemos clases java que tienen dependencias de otras clases muy complejas y queremos hacer test de Junit de ellas, podemos encontrarnos con que tenemos que hacer todo un montaje para poder construir y que funcione la clase que queremos testear. Aquí es donde Mockito vienen en nuestra ayuda. Mockito puede simular esas clases complejas que necesita nuestra clase para que podamos construirla sin tanto montaje.

```
@ExtendWith(MockitoExtension.class)
class LibroServiceTest {
   @Mock
   LibroRepository libroRepository;

   @InjectMocks
   LibroServiceImpl servicio;
}
```

Primero, nuestra clase de test la anotamos con:

- **@RunWith(MockitoJUnitRunner.class)**: si estamos con JUnit 4 o con JUnit 5 usando *junit-vintage-engine*
- **@ExtendWith(MockitoExtension.class)**: si estamos con JUnit 5 y *junit-jupiter-engine*.

Esto hace que mockito entre en acción para este test. Si no lo ponemos, no se hará caso de todo lo que pongamos de mockito en el test.

Las clases de las que queramos mock object las anotamos como **@Mock**. Esto hará que mockito genere automáticamente para ellas mock objects que podemos configurar durante el test. No es necesario que hagamos new ni que escribamos nada de código.

La clase de la que queremos hacer el test la anotamos con **@InjectMocks**. Esto hará que mockito directamente la instancie y le pase los mock object. Mockito es listo, busca los constructores que admitan esas clases pasa pasárselos o los atributos dentro de la clase para darles valor.

```
@Test
void getLibroTest(){
    Libro libroEsperado = new Libro(1L, "titulo1", "autor1");
    Mockito.when(libroRepository.findById(1L)).thenReturn(Optional.of(libroEsperado));
    Optional<Libro> resultado = servicio.getLibro(1L);
    if (resultado.isPresent()){
        assertEquals(libroEsperado, resultado.get());
    }
   Mockito.verify(libroRepository, Mockito.times(2)).findById(1L);
}
```

Los objectos simulados (**mocks**) pueden devolver diferentes valores según los argumentos pasados a un método. La cadena de métodos **when(….).thenReturn(….)** se utiliza para especificar un valor de retorno para una llamada de método con parámetros predefinidos.

También puedes usar métodos como **anyString()**, **anyInt()** o **any(Object)** para definir que, según el tipo de entrada, se debe devolver un determinado valor.

Si especifica más de un valor, se devuelven en el orden de especificación, hasta que se utiliza el último. Posteriormente se devuelve el último valor especificado.

Nos metemos ya en el test. Lo primero es configurar los mock para que devuelvan lo que queramos. La línea

Mockito.when(libroRepository.findById(1L)).thenReturn(Optional.of(libroEsperado));

le está diciendo a Mockito que cuando se llame al método *libroRepository.findById(1L)*, entonces debe devolver el objeto opcional *libroEsperado*. La sintaxis es bastante intuitiva.

Luego, nuestro test, ya puede llamar a la clase bajo test y a su método *servicio.getLibro(1L)*.

¿Cómo miramos ahora el resultado?

Además de los **asserts**, podemos **verificar** que el objeto simulado llama a un método una sola vez. También podemos especificar que el método se ejecute más de una vez.

En el ejemplo: *Mockito.verify(libroRepository, Mockito.times(2)).findById(1L);*

estaríamos verificando que libroRepository (el objeto simulado) llame al método findById(1L) dos veces.