**Junit 5 and Mockito:**

**Junit:**

JUnit is a Java unit testing framework that’s one of the best test methods for regression testing. An open-source framework, it is used to write and run repeatable automated tests.

**Mockito:**

Mockito is a mocking framework, JAVA-based library that is used for effective unit testing of JAVA applications. Mockito is used to mock interfaces so that a dummy functionality can be added to a mock interface that can be used in unit testing.

To test Spring Boot Application, community provided **spring-boot-starter-test** dependency.

The spring-boot-starter-test dependency includes [**JUnit 5** dependencies, and **Mockito**. So, we only need to include this dependency](https://howtodoinjava.com/junit5/junit5-maven-dependency/). Below is the maven dependency required for unit testing spring boot application.

**<dependency>**

**<groupId>org.springframework.boot</groupId>**

**<artifactId>spring-boot-starter-test</artifactId>**

**<scope>test</scope>**

**</dependency>**

Below dependencies are provided with spring-boot-starter-test:

Text

Description automatically generated

For Testing the services, we can use **@SpringBootTest** or **@ExtendWith** annotations on top of each test class.

* **@ExtendWith** is a [Junit](https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/lang/annotation/Repeatable.html) 5  annotation, that is used to register [extensions](https://junit.org/junit5/docs/5.8.0/api/org.junit.jupiter.api/org/junit/jupiter/api/extension/Extension.html) for the annotated test class, test interface, test method, parameter, or field.
* Annotated parameters are supported in test class constructors, in test methods, and in **@BeforeAll, @AfterAll, @BeforeEach, and @AfterEach** lifecycle methods.
* @ExtendWith fields may be either static or non-static.
* @ExtendWith accepts multiple extensions. In that, @ExtendWith(**MocitoExtensions**.class) and @ExtendWith(**SpringExtensions**.class) are used mostly.
* **SpringExtension** integrates the Spring TestContext Framework into JUnit5s Jupiter programming model.
* **MocitoExtensions** JUnit Jupiter equivalent of JUnit4MockitoJUnitRunner
* **@SpringBootTest** has inherited features of both **SpringExtension** and **MocitoExtensions.**
* Hence, we can use either **@SpringBootTest** or **@ExtendWith** annotations to test our classes.

Below is the sample code snippet of both annotations,

**@ExtendWith(MocitoExtensions.class)**

Graphical user interface, text, application, email

Description automatically generated

**@ExtendWith(SpringExtensions.class)**

Graphical user interface, text, application, email

Description automatically generated

**@SpringBootTest:**

Graphical user interface, text, application, email

Description automatically generated

Junit 5 Jupiter Annotations:

* @Test – Specifies that the method is a test method
* @ParameterizedTest - make it possible to run a test multiple times with different arguments.
* @TestMethodOrder – it defines the order of test methods execution
* @TestClassOrder – it defines the order of test classes execution
* @DisaplyName – Defines the custom name for test class.
* @BeforeEach – Method annotated with this will execute each test before
* @BeforeAll – Method annotated with this will execute before all tests
* @AfterEach – Method annotated with this will execute all test before starts execution
* @AfterAll – Method annotated with this will execute after all test’s execution completed
* @ExtendWith - Extensions can be registered *declaratively* via [@ExtendWith](https://junit.org/junit5/docs/current/user-guide/#extensions-registration-declarative)
* [@RegisterExtension](https://junit.org/junit5/docs/current/api/org.junit.jupiter.api/org/junit/jupiter/api/extension/RegisterExtension.html) - Developers can register extensions *programmatically* by annotating fields in test classes with [@RegisterExtension](https://junit.org/junit5/docs/current/api/org.junit.jupiter.api/org/junit/jupiter/api/extension/RegisterExtension.html).
* @TempDir - The built-in [TempDirectory](https://github.com/junit-team/junit5/tree/r5.9.0/junit-jupiter-engine/src/main/java/org/junit/jupiter/engine/extension/TempDirectory.java) extension is used to create and clean up a temporary directory for an individual test or all tests in a test class.
* @Tag - used to filter the test cases from test suites.

Below are the code examples for above annotations,

**@Test, @DisaplyName, @ParameterizedTest and @Valuesource:**

Graphical user interface, application

Description automatically generated

**@TestMethodOrder:**

Graphical user interface, text, application

Description automatically generated

**@TempDir**

Graphical user interface, text, application

Description automatically generated

**@BeforeEach:**

Text

Description automatically generated

**@Tag**

Graphical user interface, text, application, email

Description automatically generated

**Mockito Annotations:**

* The **@Mock** annotation creates a mock implementation for the class it is annotated with.
* Unlike @Mock, the **@Spy** creates the real time instance for the class
* **@InjectMocks** also creates the mock implementation and additionally injects the dependent mocks that are marked with the annotations **@Mock** into it.
* The **@Captor** annotation is used to create an ArgumentCaptor instance which is used to **capture method argument values** for further assertions.

Below are the code examples for Mockito annotations mentioned above,

**@Mock and @InjectMocks:**

Graphical user interface, text, application

Description automatically generated

**@Spy:**



**@Cpator:**

Graphical user interface, application

Description automatically generated

* **Mockito.verify()** method can be used to test number of method invocations. We can test exact number of times, at least once, at least, at most number of invocation times for a mocked method.
* **doReturn vs thenReturn**: used to stub the actual call while running the test.

**Example:**

**User user = Mockito.mock(User.class);**

**when(user.getName()).thenReturn("John");** - type chcking happens

**doReturn(true).when(user).getName()); -** no type checking

* Assertions are used to compare expected and actual result. Based on the assertions result the test case will success or fail. In the screenshots attached used junit 5 Assertions for validating the actual and expected results of the testcases. Below are the some annotations,
  + **assertArrayEquals** compares the contents of an actual array to an expected array.
  + **assertEquals** compares an actual value to an expected value.
  + **assertNotEquals** compares two values to validate that they are not equal.
  + **assertTrue** validates that the provided value is true.
  + **assertFalse** validates that the provided value is false.
  + **assertLinesMatch** compares two lists of Strings.
  + **assertNull** validates that the provided value is null.
  + **assertNotNull** validates that the provided value is not null.
  + **assertSame** validates that two values reference the same object.
  + **assertNotSame** validates that two values do not reference the same object.
  + **assertThrows** validates that the execution of a method throws an expected exception

Exceptions Testing can be done using **assertThrows** like below:

Graphical user interface, application

Description automatically generated

* Private method is not possible to call using Mockito. We can write testcases for private methods using below code,

Graphical user interface, text, application

Description automatically generated

* For mocking static methods, mockStatic will be used. For this to work, below dependency is required to add,

**<dependency>**

**<groupId>org.mockito</groupId>**

**<artifactId>mockito-inline</artifactId>**

**<scope>test</scope>**

**</dependency>**

Below is the code snipped for static methods mocking,

Graphical user interface, text, application

Description automatically generated

**Rest API Testing:**

1. We can use **@MockMvc** to test the api calls.
2. Mock.perform() is used to call post or get or any other api calls by passing the required parameter and inputs.
3. Basic information needs to be provided while calling api is content type, content, and headers.

Below is the code snippet for rest api calls,

Graphical user interface, text, application, email

Description automatically generated

**Reference Links:**

<https://www.lambdatest.com/blog/junit5-mockito-tutorial/>

<https://levelup.gitconnected.com/unit-test-with-junit5-and-mockito-6935431f5d7c>

<https://javadoc.io/doc/org.mockito/mockito-core/3.0.0/org/mockito/Mockito.html>

<https://howtodoinjava.com/junit5/junit-5-assertions-examples/> - Assertions Info

<http://sangsoonam.github.io/2019/02/04/mockito-doreturn-vs-thenreturn.html>