Unit testing is a testing approach that lets the developers themselves test the individual classes to check for issues .

Note: To mock properties file and static member we need to work with powerMockito framework .

Dependency in pom.xml =

spring-boot-starter-test : this dependency makes a list of annotation available at the code level to make the unit testing extremely simple .

Annotaiton used :

1. @RunWith(SpringRunner.class) = This is class level annotation that informs JUNIT to run using Spring’s support for the test .
2. @WebMvcTest(value=CustomerController.class)

This is a class level annotation that helps writing test cases for Spring Controller . This annotation can be fed with the controller name that should be launched for unit testing so that other controllers will not come into picuture when this unit test is executed

1. @MockMvc = it is used to test the weblayer of an application . MockMvc handles the incoming HTTP request and hands it off to the controller without the server being started . SO it is preferable to test the controllers with MockMvc . but we need to autowire MockMvc in test class .
2. @MockBean : This annotation helps in adding mock objects to the spring application context . Mock will create a dummy of an existing object and will replace the same in application context . Mocking is quite useful when the bean that is required by a unit test case is not ready or unstable .
3. @Before : The method annoted with this will execute before the test methods get executed .
4. @Test : The method marked with this as a test method .
5. @Stub : Stubs are the objects that holds predefined data and uses it to give response during tests . A stab is an object that resembles a real object with the minimum number of methods needed to a test .

* Used for state verification
* Returns the predefined output regardless of the input .

1. @Spy : = Spies are knows as partially mock object . it creates a partial object or half dommy or the real object by stubbing or spying the real ones .
2. @Mock : = It gives full control over the mocked object’s behaviour . Mock are the object that store method call . It is created using a library or mocking framework like Mockito, jmock,EasyMock
3. @InjectMock : It will inject mocked object into other object . We can say it will do the dependency injection .
4. @ExtendsWith(MockitoExtension.class) :

Code example :

@RunWith(SpringRunner.class)

@WebMvcTest(value=customerController.class)

**Public class CustomerControllerTest**{

@AutoWirred

**Private MockMvc mockmvc;**

@MockBean

**Private CusotmerService customerService;**

@Before

Public void beforeTest()

{ //code before test method execution goes here

}

@Test

Public void controllerTest()

{ // code here to be executed }

}

**Mostly used method in Junit Test ? ( https://www.journaldev.com/21681/junit-assertions)**

1. **Fail() :** This is used to fail a test, it’s useful when your test method is work in progress and you want to indicate that by fail-fast your test. There are many overloaded fail() methods, let’s look at some of them.
2. **assertNull() and asserNotNull() :** These methods are used to check if the specified object is null or not. We can also specify custom failure message.
3. **asserSame() and assertNotSame() :** These methods are used to assert that expected and actual elements are same or not. JUnit uses == operator for these methods to check if they are referring to same object or not.
4. **assertTure() and assertFalse() :** Asserts that the supplied condition is true or false.
5. **assertEquals(expected , actual) and assertNotEquals() :** These two methods are used to assert that expected and actual objects are equal or not. They use [equals()](https://www.journaldev.com/21095/java-equals-hashcode) method of the object to check equality.
6. **asserArrayEquals() :** Asserts that expected and actual arrays are deeply equal. The arrays elements are matched index by index.
7. **assertIterableEquals() :** This method is used for iterables to check if they are equal or not. Note that the underlying implementation can be different. The elements are matched index by index for equality.
8. **assertThrows() :** Asserts that execution of the supplied executable throws an exception of the expectedType and returns the exception.

If no exception is thrown, or if an exception of a different type is thrown, this method will fail.

This method follows inheritance hierarchy, so the assert will pass if the expected type is Exception and actual is RuntimeException.

1. **assertDoesNotThrow() :** Asserts that execution of the supplied executable does not throw any kind of exception. This method takes org.junit.jupiter.api.function.Executable instance as argument. Executable is a [functional interface](https://www.journaldev.com/2763/java-8-functional-interfaces) that can be used to implement any generic block of code that potentially throws a Throwable.
2. **assertAll() :** Asserts that all supplied executables do not throw exceptions. We can pass Array, [Stream](https://www.journaldev.com/2774/java-8-stream) or a Collection of Executable objects.
3. **assertTimeOut() :** Asserts that execution of the supplied executable completes before the given timeout is exceeded.