^^^

**org.apache.commons.lang3.RandomStringUtils.randomAlphabetic**

Generate random Strings

^^^

**org.assertj.core.api.Assertions.assertThat**

e.g. assertThat(added, is(false));

^^^

**org.hamcrest.Matchers.containsString**

Creates a matcher that matches if the examined String contains the specified String anywhere.

**org.hamcrest.Matchers.equalTo**

Creates a matcher that matches when the examined object is logically equal to the specified operand, as determined by calling the Object.equals(java.lang.Object) method on the examined object.

**org.hamcrest.Matchers.hasItem**

Creates a matcher for Iterables that only matches when a single pass over the examined Iterable yields at least one item that is equal to the specified item. The traversal of the examined Iterable stops when a match found.

**org.hamcrest.Matchers.is**

A shortcut to the frequently used is(equalTo(x)).

^^^

**org.junit.Assert.assertEquals**

QQQ

**org.junit.Assert.assertNotEquals**

QQQ

**org.junit.Assert.assertNull**

QQQ

**org.junit.Assert.assertThat**

assertThat(java.lang.String reason, T actual, org.hamcrest.Matcher<T> matcher)

Asserts that actual satisfies the condition specified by matcher. If not, an AssertionError is thrown with the reason and information about the matcher and failing value. E.g.:

assertThat("Help! Integers don't work", 0, is(1)); // fails:

**org.junit.Before**

QQQ

**org.junit.Rule**

Annotates fields that reference rules or methods that return a rule. A field must be public, not static, and a subtype of TestRule (preferred) or MethodRule. A method must be public, not static, and must return a

subtype of TestRule (preferred) or MethodRule.

**org.junit.Test**

QQQ

**org.junit.rules.ExpectedException**

The ExpectedException rule allows you to verify that your code throws a specific exception.

You have to add the ExpectedException rule to your test. This doesn't affect your existing tests.

**org.junit.runner.RunWith**

When a class is annotated with @RunWith or extends a class annotated with@RunWith, JUnit will invoke the class it references to run the tests in that class instead of the runner built into JUnit. We added this feature late in development. While it seems powerful, we expect the runner APIto change as we learn how people really use it.

^^^

**org.mockito.ArgumentCaptor**

Mockito verifies argument values in natural java style: by using an equals() method. This is also the recommended way of matching arguments because it makes tests clean & simple. In some situations though, it is helpful to assert on certain arguments after the actual verification.

Warning: it is recommended to use ArgumentCaptor with verification but not with stubbing.

**org.mockito.ArgumentMatchers.any**

Matches ANYTHING, including nulls and varargs.

**org.mockito.ArgumentMatchers.anyString**

Any NON-NULL String.

Since Mockito 2.1.0, only allow non-null String.

As this is a nullable reference, the suggested API to match null wrapper would be isNull(). We felt this change would make tests harness much safer that it was with Mockito 1.x.

**org.mockito.Captor**

Captor annotation allows shorthand ArgumentCaptor creation on fields. One of the advantages of using @Captor annotation is that you can avoid warnings related capturing complex generic types. The Captor annotation is defined as below:

@Retention(value=RUNTIME)

@Target(value=FIELD)

@Documented

public @interface Captor

**org.mockito.InOrder**

Allows verification in order. E.g:

InOrder inOrder = inOrder(firstMock, secondMock);

inOrder.verify(firstMock).add("was called first");

inOrder.verify(secondMock).add("was called second");

**org.mockito.Matchers.anyInt**

Any int, Integer or null.

This method \*don't do any type checks\*, it is only there to avoid casting in your code. This might however change (type checks could be added) in a future major release.

**org.mockito.Matchers.anyString**

Any String or null.

This method \*don't do any type checks\*, it is only there to avoid casting in your code. This might however change (type checks could be added) in a future major release.

**org.mockito.Mock**

QQQ

**org.mockito.MockSettings**

Allows mock creation with additional mock settings.

Don't use it too often. Consider writing simple tests that use simple mocks.

If you cannot write a test in a simple way - refactor the code under test.

**org.mockito.Mockito.doAnswer**

Use doAnswer() when you want to stub a void method with generic Answer. Stubbing voids requires different approach from when(Object) because the compiler does not like void methods inside brackets...

**org.mockito.Mockito.doThrow**

Use doThrow() when you want to stub the void method with an exception.

**org.mockito.Mockito.mock**

Creates mock object of given class or interface.

**org.mockito.Mockito.spy**

Creates a spy of the real object. The spy calls real methods unless they are stubbed. Real spies should be used carefully and occasionally, for example when dealing with legacy code.

**org.mockito.Mockito.verify**

Verifies certain behavior HAPPENED ONCE.

**org.mockito.Mockito.when**

Enables stubbing methods. Use it when you want the mock to return particular value when particular method is called: "When the x method is called then return y".

**org.mockito.Mockito**

**org.mockito.MockitoAnnotations**

MockitoAnnotations.initMocks(this); Initializes fields annotated with Mockito annotations.

**org.mockito.exceptions.verification.NoInteractionsWanted**

No interactions wanted. See exception's cause for location of undesired invocation.

**org.mockito.exceptions.verification.TooLittleActualInvocations**

?

**org.mockito.invocation.InvocationOnMock**

A placeholder for mock, the method that was called and the arguments that were passed.

**org.mockito.junit.MockitoJUnit**

The JUnit rule can be used instead of MockitoJUnitRunner.

**org.mockito.junit.MockitoJUnitRunner**

Mockito JUnit Runner keeps tests clean and improves debugging experience. Make sure to try out MockitoJUnitRunner.StrictStubs which automatically detects stubbing argument mismatches and is planned to be the default in Mockito v3. Runner is completely optional - there are other ways you can get @Mock working, for example by writing a base class.

**org.mockito.junit.MockitoRule**

The JUnit rule can be used instead of MockitoJUnitRunner. It requires JUnit at least 4.7. This rule adds following behavior:

Initializes mocks annotated with Mock – makes MockitoAnnotations.initMocks(Object) optional

Mocks are initialized before each test method.

Validates framework usage after each test method.

See javadoc for Mockito.validateMockitoUsage().

**org.mockito.stubbing.Answer**

Generic interface to be used for configuring mock's answer. Answer specifies an action that is executed and a return value that is returned when you interact with the mock.