Testing Framework for Java

To perform unit testing, we need to create test cases. The **unit test case** is a code which ensures that the program logic works as expected.

## **Types of unit testing**

There are two ways to perform unit testing: 1) manual testing 2) automated testing.

#### **1) Manual Testing**

If you execute the test cases manually without any tool support, it is known as manual testing. It is time consuming and less reliable.

#### **2) Automated Testing**

If you execute the test cases by tool support, it is known as automated testing. It is fast and more reliable.

#### **Annotations for Junit testing**

The Junit 4.x framework is annotation based, so let's see the annotations that can be used while writing the test cases.

**@Test** annotation specifies that method is the test method.

**@Test(timeout=1000)** annotation specifies that method will be failed if it takes longer than 1000 milliseconds (1 second).

**@BeforeClass** annotation specifies that method will be invoked only once, before starting all the tests.

**@Before** annotation specifies that method will be invoked before each test.

**@After** annotation specifies that method will be invoked after each test.

**@AfterClass** annotation specifies that method will be invoked only once, after finishing all the tests.

Example:

Here’re some basic JUnit annotations you should understand:

1. @BeforeClass – Run once before any of the test methods in the class, public static void
2. @AfterClass – Run once after all the tests in the class have been run, public static void
3. @Before – Run before @Test, public void
4. @After – Run after @Test, public void
5. @Test – This is the test method to run, public void

*P.S Tested with JUnit 4.12*

BasicAnnotationTest.java

import org.junit.\*;

public class BasicAnnotationTest {

*// Run once, e.g. Database connection, connection pool*

@BeforeClass

public static void runOnceBeforeClass() {

System.out.println("@BeforeClass - runOnceBeforeClass");

}

*// Run once, e.g close connection, cleanup*

@AfterClass

public static void runOnceAfterClass() {

System.out.println("@AfterClass - runOnceAfterClass");

}

*// Should rename to @BeforeTestMethod*

*// e.g. Creating an similar object and share for all @Test*

@Before

public void runBeforeTestMethod() {

System.out.println("@Before - runBeforeTestMethod");

}

*// Should rename to @AfterTestMethod*

@After

public void runAfterTestMethod() {

System.out.println("@After - runAfterTestMethod");

}

@Test

public void test\_method\_1() {

System.out.println("@Test - test\_method\_1");

}

@Test

public void test\_method\_2() {

System.out.println("@Test - test\_method\_2");

}

}

## **Assert class**

The org.junit.Assert class provides methods to assert the program logic.

#### Methods of Assert class

The common methods of Assert class are as follows:

1. **void assertEquals(boolean expected,boolean actual)**: checks that two primitives/objects are equal. It is overloaded.
2. **void assertTrue(boolean condition)**: checks that a condition is true.
3. **void assertFalse(boolean condition)**: checks that a condition is false.
4. **void assertNull(Object obj)**: checks that object is null.
5. **void assertNotNull(Object obj)**: checks that object is not null.
6. **package** com.javatpoint.logic;
7. **public** **class** Calculation {
9. **public** **static** **int** findMax(**int** arr[]){
10. **int** max=0;
11. **for**(**int** i=1;i<arr.length;i++){
12. **if**(max<arr[i])
13. max=arr[i];
14. }
15. **return** max;
16. }
17. }

#### **Write the test case**

Here, we are using JUnit 4, so there is no need to inherit TestCase class. The main testing code is written in the testFindMax() method. But we can also perform some task before and after each test, as you can see in the given program.

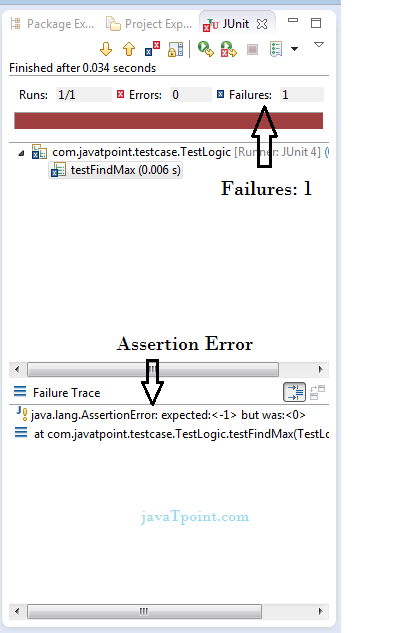
1. **package** com.javatpoint.testcase;
3. **import** **static** org.junit.Assert.\*;
4. **import** com.javatpoint.logic.\*;
5. **import** org.junit.Test;
7. **public** **class** TestLogic {
9. @Test
10. **public** **void** testFindMax(){
11. assertEquals(4,Calculation.findMax(**new** **int**[]{1,3,4,2}));
12. assertEquals(-1,Calculation.findMax(**new** **int**[]{-12,-1,-3,-4,-2}));
13. }
14. }

[download this example](https://www.javatpoint.com/src/junit/junittesting.zip)

To run this example, **right click on TestLogic class -> Run As -> 1Junit Test**.

**Output:**Assertion Error

Let's see the output displayed in eclipse IDE.



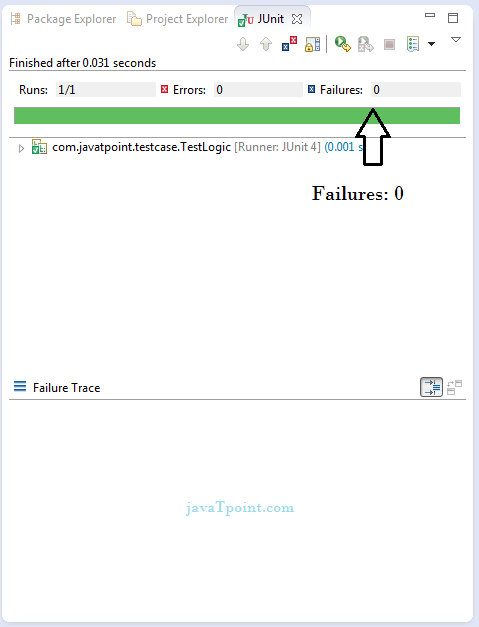
As you can see, when we pass the negative values, it throws AssertionError because second time findMax() method returns 0 instead of -1. It means our program logic is incorrect.

#### **Correct program logic**

As you can see, program logic to find the maximum number for the given array is not correct because it doesn't return -1 in case of negative values. The correct program logic is given below:

1. **package** com.javatpoint.logic;
2. **public** **class** Calculation {
4. **public** **static** **int** findMax(**int** arr[]){
5. **int** max=arr[0];//arr[0] instead of 0
6. **for**(**int** i=1;i<arr.length;i++){
7. **if**(max<arr[i])
8. max=arr[i];
9. }
10. **return** max;
11. }
12. }

If you run the junit program again, you will see the following output.



## **Another example of Junit framework**

#### **Write the program code**

1. **package** com.javatpoint.logic;
2. **public** **class** Calculation {
3. //method that returns maximum number
4. **public** **static** **int** findMax(**int** arr[]){
5. **int** max=0;
6. **for**(**int** i=1;i<arr.length;i++){
7. **if**(max<arr[i])
8. max=arr[i];
9. }
10. **return** max;
11. }
12. //method that returns cube of the given number
13. **public** **static** **int** cube(**int** n){
14. **return** n\*n\*n;
15. }
16. //method that returns reverse words
17. **public** **static** String reverseWord(String str){
19. StringBuilder result=**new** StringBuilder();
20. StringTokenizer tokenizer=**new** StringTokenizer(str," ");
22. **while**(tokenizer.hasMoreTokens()){
23. StringBuilder sb=**new** StringBuilder();
24. sb.append(tokenizer.nextToken());
25. sb.reverse();
27. result.append(sb);
28. result.append(" ");
29. }
30. **return** result.toString();
31. }
32. }

#### **Write the test case**

1. **package** com.javatpoint.testcase;
3. **import** **static** org.junit.Assert.assertEquals;
4. **import** org.junit.After;
5. **import** org.junit.AfterClass;
6. **import** org.junit.Before;
7. **import** org.junit.BeforeClass;
8. **import** org.junit.Test;
9. **import** com.javatpoint.logic.Calculation;
11. **public** **class** TestCase2 {
13. @BeforeClass
14. **public** **static** **void** setUpBeforeClass() **throws** Exception {
15. System.out.println("before class");
16. }
17. @Before
18. **public** **void** setUp() **throws** Exception {
19. System.out.println("before");
20. }
22. @Test
23. **public** **void** testFindMax(){
24. System.out.println("test case find max");
25. assertEquals(4,Calculation.findMax(**new** **int**[]{1,3,4,2}));
26. assertEquals(-2,Calculation.findMax(**new** **int**[]{-12,-3,-4,-2}));
27. }
28. @Test
29. **public** **void** testCube(){
30. System.out.println("test case cube");
31. assertEquals(27,Calculation.cube(3));
32. }
33. @Test
34. **public** **void** testReverseWord(){
35. System.out.println("test case reverse word");
36. assertEquals("ym eman si nahk",Calculation.reverseWord("my name is khan");
37. }
38. @After
39. **public** **void** tearDown() **throws** Exception {
40. System.out.println("after");
41. }
43. @AfterClass
44. **public** **static** **void** tearDownAfterClass() **throws** Exception {
45. System.out.println("after class");
46. }
48. }

**Output:**before class

before

test case find max

after

before

test case cube

after

before

test case reverse word

after

after class

# JUnit 5 tutorial - Learn how to write unit tests

## [1. Overview](https://www.vogella.com/tutorials/JUnit/article.html#overview)

JUnit is a popular unit-testing framework in the Java ecosystem. JUnit 5 added many new features based on the Java 8 version of the language.

This guide gives an introduction into unit testing with the JUnit framework using JUnit 5. It focus on the usage of the framework.

### [1.1. Configuration for using JUnit 5](https://www.vogella.com/tutorials/JUnit/article.html#junitsetup)

To use JUnit 5 you have to make the libraries available for your test code. Jump to the section which is relevant to you, for example read the Maven part, if you are using Maven as build system.

Configure Maven to use JUnit 5

Configure Gradle to use JUnit 5

Configure JUnit 5 with Eclipse

### [1.2. How to define a test in JUnit?](https://www.vogella.com/tutorials/JUnit/article.html#unittesting_junit_test)

A JUnit test is a method contained in a class which is only used for testing. This is called a Test class. To mark a method as a test method, annotate it with the @Test annotation. This method executes the code under test.

The following code defines a minimal test class with one minimal test method.

**package** **com.vogella.junit.first;**

**import** **static** org**.**junit**.**jupiter**.**api**.**Assertions**.**assertTrue**;**

**import** **org.junit.jupiter.api.Test;**

**class** **AClassWithOneJUnitTest** **{**

**@Test**

**void** demoTestMethod**()** **{**

assertTrue**(true);**

**}**

**}**

You can use assert methods, provided by JUnit or another assert framework, to check an expected result versus the actual result. Such statement are called asserts or assert statements.

Assert statements typically allow to define messages which are shown if the test fails. You should provide here meaningful messages to make it easier for the user to identify and fix the problem. This is especially true if someone looks at the problem, who did not write the code under test or the test code.

### [1.3. Example for developing a JUnit 5 test for another class](https://www.vogella.com/tutorials/JUnit/article.html#unittesting_junitexample)

The following example defines a Java class and defines software tests for it.

Assume you have the following class which you want to test.

**package** **com.vogella.junit5;**

**public** **class** **Calculator** **{**

**public** **int** multiply**(int** a**,** **int** b**)** **{**

**return** a **\*** b**;**

**}**

**}**

A test class for the above class could look like the following.

**package** **com.vogella.junit5;**

**import** **static** org**.**junit**.**jupiter**.**api**.**Assertions**.**assertEquals**;**

**import** **org.junit.jupiter.api.BeforeEach;**

**import** **org.junit.jupiter.api.DisplayName;**

**import** **org.junit.jupiter.api.RepeatedTest;**

**import** **org.junit.jupiter.api.Test;**

**class** **CalculatorTest** **{**

**Calculator** calculator**;**

**@BeforeEach**

**void** setUp**()** **{**

calculator **=** **new** **Calculator();**

**}**

**@Test**

**@DisplayName("Simple multiplication should work")**

**void** testMultiply**()** **{**

assertEquals**(20,** calculator**.**multiply**(4,** **5),**

**"Regular multiplication should work");**

**}**

**@RepeatedTest(5)**

**@DisplayName("Ensure correct handling of zero")**

**void** testMultiplyWithZero**()** **{**

assertEquals**(0,** calculator**.**multiply**(0,** **5),** **"Multiple with zero should be zero");**

assertEquals**(0,** calculator**.**multiply**(5,** **0),** **"Multiple with zero should be zero");**

**}**

**}**

|  |  |
| --- | --- |
|  | The method annotated with @BeforeEach runs before each test |
|  | A method annotated with @Test defines a test method |
|  | @DisplayName can be used to define the name of the test which is displayed to the user |
|  | This is an assert statement which validates that expected and actual value is the same, if not the message at the end of the method is shown |
|  | @RepeatedTest defines that this test method will be executed multiple times, in this example 5 times |

### [1.4. JUnit test class naming conventions](https://www.vogella.com/tutorials/JUnit/article.html#junit_namingconventions_maven)

Build tools like Maven use a pattern to decide if a class is a test classes or not. The following is the list of classes Maven considers automatically during its build:

\*\*/Test\*.java

\*\*/\*Test.java

\*\*/\*Tests.java

\*\*/\*TestCase.java

|  |  |
| --- | --- |
|  | includes all of its subdirectories and all Java filenames that start with Test. |
|  | includes all of its subdirectories and all Java filenames that end with Test. |
|  | includes all of its subdirectories and all Java filenames that end with Tests. |
|  | includes all of its subdirectories and all Java filenames that end with TestCase. |

Therefore, it is common practice to use the Test or Tests suffix at the end of test classes names.

### [1.5. Where should the test be located?](https://www.vogella.com/tutorials/JUnit/article.html#junit_testorganization)

Typical, unit tests are created in a separate source folder to keep the test code separate from the real code. The standard convention from the Maven and Gradle build tools is to use:

* src/main/java - for Java classes
* src/test/java - for test classes

### [1.6. Static imports and unit testing](https://www.vogella.com/tutorials/JUnit/article.html#usingjunit_staticimports)

JUnit 5 allows to use static imports for its assertStatements to make the test code short and easy to read. Static imports are a Java feature that allows fields and methods defined in a class as public static to be used without specifying the class in which the field is defined.

JUnit assert statements are typically defined as public static to allow the developer to write short test statements. The following snippet demonstrates an assert statement with and without static imports.

*// without static imports you have to write the following statement*

**import** **org.junit.jupiter.api.Assertions;**

*// more code*

**Assert.**assertEquals**("10 x 5 must be 50",** **50,** tester**.**multiply**(10,** **5));**

*// alternatively define assertEquals as static import*

**import** **static** org**.**junit**.**jupiter**.**api**.**Assertions**.**assertEquals**;**

*// more code*

*// use assertEquals directly because of the static import*

assertEquals**(**calculator**.**multiply**(4,5),** **20,** **"Regular multiplication should work");**

## [2. Assertions and assumptions](https://www.vogella.com/tutorials/JUnit/article.html#assertions-and-assumptions)

JUnit 5 comes with multiple assert statements, which allows you to test your code under test. Simple assert statements like the following allow to check for true, false or equality. All of them are static methods from the org.junit.jupiter.api.Assertions.\* package.

| **Assert statement** | **Example** |
| --- | --- |
| assertEquals | assertEquals(4, calculator.multiply(2, 2),"optional failure message"); |
| assertTrue | assertTrue('a' < 'b', () → "optional failure message"); |
| assertFalse | assertFalse('a' > 'b', () → "optional failure message"); |
| assertNotNull | assertNotNull(yourObject, "optional failure message"); |
| assertNull | assertNull(yourObject, "optional failure message"); |

Messages can be created via lambda expressions, to avoid the overhead in case the construction of the message is expensive.

assertTrue**('a'** **<** **'b',** **()** **->** **"Assertion messages can be lazily evaluated -- "**

**+** **"to avoid constructing complex messages unnecessarily.");**

### [2.1. Testing for exceptions](https://www.vogella.com/tutorials/JUnit/article.html#testing-for-exceptions)

Testing that certain exceptions are thrown are be done with the org.junit.jupiter.api.Assertions.expectThrows() assert statement. You define the expected Exception class and provide code that should throw the exception.

**import** **static** org**.**junit**.**jupiter**.**api**.**Assertions**.**assertThrows**;**

**@Test**

**void** exceptionTesting**()** **{**

*// set up user*

**Throwable** exception **=** assertThrows**(IllegalArgumentException.**class**,** **()** **->** user**.**setAge**("23"));**

assertEquals**("Age must be an Integer.",** exception**.**getMessage**());**

**}**

This lets you define which part of the test should throw the exception. The test will still fail if an exception is thrown outside of this scope.

### [2.2. Testing multiple assertions (grouped assertions) with assertAll](https://www.vogella.com/tutorials/JUnit/article.html#testing-multiple-assertions-grouped-assertions-with-assertall)

If an assert fails in a test, JUnit will stop executing the test and additional asserts are not checked. In case you want to ensure that all asserts are checked you can assertAll.

In this grouped assertion all assertions are executed, even after a failure. The error messages get also grouped together.

**@Test**

**void** groupedAssertions**()** **{**

**Address** address **=** **new** **Address();**

assertAll**("address name",**

**()** **->** assertEquals**("John",** address**.**getFirstName**()),**

**()** **->** assertEquals**("User",** address**.**getLastName**())**

**);**

**}**

If these tests fail, the result looks like the following:

=> org.opentest4j.MultipleFailuresError: address name (2 failures)

expected: <John> but was: <null>

expected: <User> but was: <null>

### [2.3. Defining timeouts in your tests](https://www.vogella.com/tutorials/JUnit/article.html#defining-timeouts-in-your-tests)

If you want to ensure that a test fails, if it isn’t done in a certain amount of time you can use the assertTimeout() method. This assert fails the method if the timeout is exceeded.

**import** **static** org**.**junit**.**jupiter**.**api**.**Assertions**.**assertTimeout**;**

**import** **static** java**.**time**.**Duration**.**ofSeconds**;**

**import** **static** java**.**time**.**Duration**.**ofMinutes**;**

**@Test**

**void** timeoutNotExceeded**()** **{**

assertTimeout**(**ofMinutes**(1),** **()** **->** service**.**doBackup**());**

**}**

*// if you have to check a return value*

**@Test**

**void** timeoutNotExceededWithResult**()** **{**

**String** actualResult **=** assertTimeout**(**ofSeconds**(1),** **()** **->** **{**

**return** restService**.**request**(**request**);**

**});**

assertEquals**(200,** request**.**getStatus**());**

**}**

**=>** org**.**opentest4j**.**AssertionFailedError**:** execution exceeded timeout of **1000** ms by **212** ms

If you want your tests to cancel after the timeout period is passed you can use the assertTimeoutPreemptively() method.

**@Test**

**void** timeoutNotExceededWithResult**()** **{**

**String** actualResult **=** assertTimeoutPreemptively**(**ofSeconds**(1),** **()** **->** **{**

**return** restService**.**request**(**request**);**

**});**

assertEquals**(200,** request**.**getStatus**());**

**}**

**=>** org**.**opentest4j**.**AssertionFailedError**:** execution timed out after **1000** ms

|  |  |
| --- | --- |
|  | Such a test might be flacky, in case the test server is busy, the test execution might take longer and therefore such a test might fails from time to time. |

### [2.4. How to disable tests](https://www.vogella.com/tutorials/JUnit/article.html#how-to-disable-tests)

The @Disabled or @Disabled("Why disabled") annotation marks a test to be disabled. This is useful when the underlying code has been changed and the test case has not yet been adapted of if the test demonstrates an incorrect behavior in the code which has not yet been fixed. It is best practice to provide the optional description, why the test is disabled.

Alternatively you can use Assumptions.assumeFalse or Assumptions.assumeTrue to define a condition for test execution. Assumptions.assumeFalse marks the test as invalid, if its condition evaluates to true. Assumptions.assumeTrue evaluates the test as invalid if its condition evaluates to false. For example, the following disables a test on Linux:

**Assumptions.**assumeFalse**(System.**getProperty**("os.name").**contains**("Linux"));**

This gives TestAbortedException which the test runners evaluate as skipped tests.

For example the following testMultiplyWithZero is skipped if executed on Linux.

**package** **com.vogella.junit5;**

**import** **static** org**.**junit**.**jupiter**.**api**.**Assertions**.**assertEquals**;**

**import** **org.junit.jupiter.api.Assumptions;**

**import** **org.junit.jupiter.api.BeforeEach;**

**import** **org.junit.jupiter.api.DisplayName;**

**import** **org.junit.jupiter.api.RepeatedTest;**

**import** **org.junit.jupiter.api.Test;**

**public** **class** **CalculatorTest** **{**

**private** **Calculator** calculator**;**

**@BeforeEach**

**void** setUp**()** **throws** **Exception** **{**

calculator **=** **new** **Calculator();**

**}**

**@RepeatedTest(5)**

**@DisplayName("Ensure correct handling of zero")**

**void** testMultiplyWithZero**()** **{**

**Assumptions.**assumeFalse**(System.**getProperty**("os.name").**contains**("Linux"));**

assertEquals**(**calculator**.**multiply**(0,5),** **0,** **"Multiple with zero should be zero");**

assertEquals**(**calculator**.**multiply**(5,0),** **0,** **"Multiple with zero should be zero");**

**}**

**}**

You can also write an extension for @ExtendWith which defines conditions under which a test should run.

## [3. Dynamic and parameterized tests](https://www.vogella.com/tutorials/JUnit/article.html#dynamic-and-parameterized-tests)

JUnit 5 supports the creation of dynamic tests via code. You can also run tests with a set of different input values with parameterized tests.

Both approaches are described here.

### [3.1. Using Dynamic Tests](https://www.vogella.com/tutorials/JUnit/article.html#dynamic_tests)

Dynamic test methods are annotated with @TestFactory and allow to create multiple tests of type DynamicTest with your code. They can return:

* an Iterable
* a Collection
* a Stream

JUnit 5 creates and runs all dynamic tests during test execution.

Methods annotated with @BeforeEach and @AfterEach are not called for dynamic tests. This means, that you can’t use thesm to reset the test object, if you change it’s state in the lambda expression for a dynamic test.

In the following example we define a method to return a Stream of DynamicTest instances.

**package** **com.vogella.unittest;**

**import** **static** org**.**junit**.**jupiter**.**api**.**Assertions**.**assertEquals**;**

**import** **static** org**.**junit**.**jupiter**.**api**.**DynamicTest**.**dynamicTest**;**

**import** **java.util.Arrays;**

**import** **java.util.stream.Stream;**

**import** **org.junit.jupiter.api.DynamicTest;**

**import** **org.junit.jupiter.api.TestFactory;**

**class** **DynamicTestCreationTest** **{**

**@TestFactory**

**Stream<DynamicTest>** testDifferentMultiplyOperations**()** **{**

**MyClass** tester **=** **new** **MyClass();**

**int[][]** data **=** **new** **int[][]** **{** **{** **1,** **2,** **2** **},** **{** **5,** **3,** **15** **},** **{** **121,** **4,** **484** **}** **};**

**return** **Arrays.**stream**(**data**).**map**(**entry **->** **{**

**int** m1 **=** entry**[0];**

**int** m2 **=** entry**[1];**

**int** expected **=** entry**[2];**

**return** dynamicTest**(**m1 **+** **" \* "** **+** m2 **+** **" = "** **+** expected**,** **()** **->** **{**

assertEquals**(**expected**,** tester**.**multiply**(**m1**,** m2**));**

**});**

**});**

**}**

*// class to be tested*

**class** **MyClass** **{**

**public** **int** multiply**(int** i**,** **int** j**)** **{**

**return** i **\*** j**;**

**}**

**}**

**}**

### [3.2. Using Parameterized Tests](https://www.vogella.com/tutorials/JUnit/article.html#using-parameterized-tests)

Junit5 also supports parameterized tests. To use them you have to add the junit-jupiter-params package as a test dependencies.

Adding junit-jupiter-params dependency for a Maven build

Adding junit-jupiter-params dependency for a Gradle build

For this example we use the @MethodSource annotation.

We give it the name of the function(s) we want it to call to get it’s test data. The function has to be static and must return either a Collection, an Iterator, a Stream or an Array. On execution the test method gets called once for every entry in the data source. In contrast to [Dynamic Tests](https://www.vogella.com/tutorials/JUnit/article.html#dynamic_tests) @BeforeEach and @AfterEach methods will be called for parameterized tests.

**import** **static** org**.**junit**.**jupiter**.**api**.**Assertions**.\*;**

**import** **org.junit.jupiter.params.ParameterizedTest;**

**import** **org.junit.jupiter.params.provider.MethodSource;**

**public** **class** **UsingParameterizedTest** **{**

**public** **static** **int[][]** data**()** **{**

**return** **new** **int[][]** **{** **{** **1** **,** **2,** **2** **},** **{** **5,** **3,** **15** **},** **{** **121,** **4,** **484** **}** **};**

**}**

**@ParameterizedTest**

**@MethodSource(**value **=** **"data")**

**void** testWithStringParameter**(int[]** data**)** **{**

**MyClass** tester **=** **new** **MyClass();**

**int** m1 **=** data**[0];**

**int** m2 **=** data**[1];**

**int** expected **=** data**[2];**

assertEquals**(**expected**,** tester**.**multiply**(**m1**,** m2**));**

**}**

*// class to be tested*

**class** **MyClass** **{**

**public** **int** multiply**(int** i**,** **int** j**)** **{**

**return** i **\*** j**;**

**}**

**}**

**}**

#### [**3.2.1. Data sources**](https://www.vogella.com/tutorials/JUnit/article.html#data-sources)

The following table gives an overview of all possible test data sources for parameterized tests.

| *Table 1. Table Parameterized Tests Data Sources* | |
| --- | --- |
| **Annotation** | **Description** |
| **@ValueSource(**ints **=** **{** **1,** **2,** **3** **})** | Lets you define an array of test values. Permissible types are String, int, long, or double. |
| **@EnumSource(**value **=** **Months.**class**,** names **=** **{"JANUARY",** **"FEBRUARY"})** | Lets you pass Enum constants as test class. With the optional attribute names you can choose which constants should be used. Otherwise all attributes are used. |
| **@MethodSource(**names **=** **"genTestData")** | The result of the named method is passed as argument to the test. |
| **@CsvSource({** **"foo, 1",** **"'baz, qux', 3"** **})**  **void** testMethod**(String** first**,** **int** second**)** **{** | Expects strings to be parsed as Csv. The delimiter is ','. |
| **@ArgumentsSource(MyArgumentsProvider.**class**)** | Specifies a class that provides the test data. The referenced class has to implement the ArgumentsProvider interface. |

#### [**3.2.2. Argument conversion**](https://www.vogella.com/tutorials/JUnit/article.html#argument-conversion)

JUnit tries to automatically convert the source strings to match the expected arguments of the test method.

If you need explicit conversion you can specify a converter with the @ConvertWith annotation. To define your own converter you have to implement the ArgumentConverter interface. In the following example we use the abstract SimpleArgumentConverter base class.

**@ParameterizedTest**

**@ValueSource(**ints **=** **{1,** **12,** **42})**

**void** testWithExplicitArgumentConversion**(@ConvertWith(ToOctalStringArgumentConverter.**class**)** **String** argument**)** **{**

**System.**err**.**println**(**argument**);**

assertNotNull**(**argument**);**

**}**

**static** **class** **ToOctalStringArgumentConverter** **extends** **SimpleArgumentConverter** **{**

**@Override**

**protected** **Object** convert**(Object** source**,** **Class<?>** targetType**)** **{**

assertEquals**(Integer.**class**,** source**.**getClass**(),** **"Can only convert from Integers.");**

assertEquals**(String.**class**,** targetType**,** **"Can only convert to String");**

**return** **Integer.**toOctalString**((Integer)** source**);**

**}**

**}**

[ⓘ](https://www.pixfuture.com/advertisers/?id0348293521d)

### [4.2. Test execution order](https://www.vogella.com/tutorials/JUnit/article.html#test-execution-order)

JUnit runs test methods is a deterministic but unpreditable order (MethodSorters.DEFAULT). You can use the @TestMethodOrder on the class to control the execution order of the tests, via:

* @TestMethodOrder(MethodOrderer.OrderAnnotation.class) - Allows to use the @Order(int) annotation on methods to define order
* @TestMethodOrder(MethodOrderer.DisplayName.class) - runs test method in alphanumeric order of display name
* @TestMethodOrder(MethodOrderer.MethodName.class) - runs test method in alphanumeric order of method name
* Custom implementation - Implement your own MethodOrderer via the orderMethods method, which allows you to call context.getMethodDescriptors().sort(..)

The following demonstrates this with OrderAnnotation.class.

**package** **com.vogella.unittest.sortmethods;**

**import** **org.junit.jupiter.api.MethodOrderer.OrderAnnotation;**

**import** **org.junit.jupiter.api.Order;**

**import** **org.junit.jupiter.api.Test;**

**import** **org.junit.jupiter.api.TestMethodOrder;**

**@TestMethodOrder(OrderAnnotation.**class**)**

**class** **OrderAnnotationDemoTest** **{**

**@Test**

**@Order(1)**

**void** firstOne**()** **{**

*// test something here*

**}**

**@Test**

**@Order(2)**

**void** secondOne**()** **{**

*// test something here*

**}**

**}**

### [4.3. Using the @TempDir annotation to create temporary files and paths](https://www.vogella.com/tutorials/JUnit/article.html#using-the-tempdir-annotation-to-create-temporary-files-and-paths)

The @TempDir annotations allows to annotate non-private fields or method parameters in a test method of type Path or File. JUnit 5 has registered a `ParameterResolutionException for this annotation and will create temporary files and paths for the tests. It will also remove the temporary files are each test.

**@Test**

**@DisplayName("Ensure that two temporary directories with same files names and content have same hash")**

**void** hashTwoDynamicDirectoryWhichHaveSameContent**(@TempDir** **Path** tempDir**,** **@TempDir** **Path** tempDir2**)** **throws** **IOException** **{**

**Path** file1 **=** tempDir**.**resolve**("myfile.txt");**

**List<String>** input **=** **Arrays.**asList**("input1",** **"input2",** **"input3");**

**Files.**write**(**file1**,** input**);**

assertTrue**(Files.**exists**(**file1**),** **"File should exist");**

**Path** file2 **=** tempDir2**.**resolve**("myfile.txt");**

**Files.**write**(**file2**,** input**);**

assertTrue**(Files.**exists**(**file2**),** **"File should exist");**

**}**

### [4.4. Test Suites](https://www.vogella.com/tutorials/JUnit/article.html#test-suites)

The 5.8 release of JUnit 5 is planned to have test suite support included.

**import** **org.junit.platform.suite.api.SelectPackages;**

**import** **org.junit.platform.suite.api.Suite;**

**import** **org.junit.platform.suite.api.SuiteDisplayName;**

**@Suite**

**@SuiteDisplayName("JUnit Platform Suite Demo")**

**@SelectPackages("example")**

**public** **class** **SuiteDemo** **{**

**}**

## [5. Exercise: Writing a JUnit 5 test with Maven and Eclipse in 5 mins](https://www.vogella.com/tutorials/JUnit/article.html#junitmaveneclipse)

In this exercise you learn you to write a JUnit5 test using Maven and the Eclipse IDE.

### [5.1. Project creation](https://www.vogella.com/tutorials/JUnit/article.html#project-creation)

Create a new Maven project with the following settings:

* Group: com.vogella
* Artifact: com.vogella.junit.first
* Version: 0.0.1-SNAPSHOT
* Packaging: jar

### [5.2. Configure Maven dependencies for JUnit 5](https://www.vogella.com/tutorials/JUnit/article.html#configure-maven-dependencies-for-junit-5)

#### [**5.2.1. Steps required to configure Maven to use JUnit5**](https://www.vogella.com/tutorials/JUnit/article.html#steps-required-to-configure-maven-to-use-junit5)

To use JUnit5 in an Maven project, you need to:

* Configure to use Java 11 or higher, as this is required by JUnit5
* Configure the maven-surefire-plugin and maven-failsafe-plugin to be at version 2.22.2 so that they can run JUnit5
* Add dependencies to the JUnit5 API and engine for your test code

#### [**5.2.2. Configure Maven**](https://www.vogella.com/tutorials/JUnit/article.html#configure-maven)

Therefore you need to adjust your pom file, similar to the following:

**<properties>**

**<project.build.sourceEncoding>**UTF-8**</project.build.sourceEncoding>**

**<maven.compiler.source>**11**</maven.compiler.source>**

**<maven.compiler.target>**11**</maven.compiler.target>**

**</properties>**

*<!--1 -->*

**<build>**

**<plugins>**

**<plugin>**

**<artifactId>**maven-surefire-plugin**</artifactId>**

**<version>**2.22.2**</version>**

**</plugin>**

**<plugin>**

**<artifactId>**maven-failsafe-plugin**</artifactId>**

**<version>**2.22.2**</version>**

**</plugin>**

**</plugins>**

**</build>**

*<!--2 -->*

**<dependencies>**

*<!-- https://mvnrepository.com/artifact/org.hamcrest/hamcrest-library -->*

**<dependency>**

**<groupId>**org.junit.jupiter**</groupId>**

**<artifactId>**junit-jupiter-api**</artifactId>**

**<version>**5.7.2**</version>**

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

**</dependency>**

**<dependency>**

**<groupId>**org.junit.jupiter**</groupId>**

**<artifactId>**junit-jupiter-engine**</artifactId>**

**<version>**5.7.2**</version>**

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

**</dependency>**

**</dependencies>**

Once you have done this, you can start using JUnit5 in your Maven project for writing unit tests.

#### [**5.2.3. Update Maven settings (in case you are using the Eclipse IDE)**](https://www.vogella.com/tutorials/JUnit/article.html#update-maven-settings-in-case-you-are-using-the-eclipse-ide)

Right-click your pom file, select **Maven**  **Update Project** and select your project. This triggers an update of your project settings and dependencies.

### [5.3. Package creation](https://www.vogella.com/tutorials/JUnit/article.html#package-creation)

Create a package named com.vogella.junit.first in the src/main/java and src/main/test folder.

### [5.4. Create a Java class](https://www.vogella.com/tutorials/JUnit/article.html#create-a-java-class)

In the src folder, create the following class in the com.vogella.junit.first package.

**package** **com.vogella.junit.first;**

**public** **class** **MyClass** **{**

*// the following is just an example*

**public** **int** multiply**(int** x**,** **int** y**)** **{**

**if** **(**x **>** **999)** **{**

**throw** **new** IllegalArgumentException**("X should be less than 1000");**

**}**

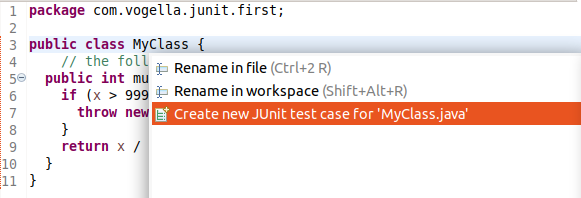
**return** x **/** y**;**

**}**

**}**

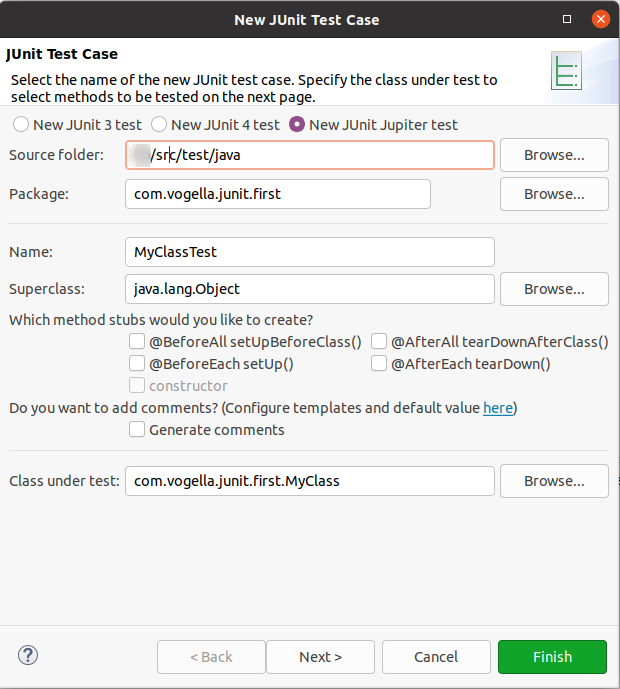
### [5.5. Create a JUnit test](https://www.vogella.com/tutorials/JUnit/article.html#create-a-junit-test)

Position the cursor on the MyClass in the Java editor and press Ctrl+1. Select that you want to create a new JUnit test from the list.

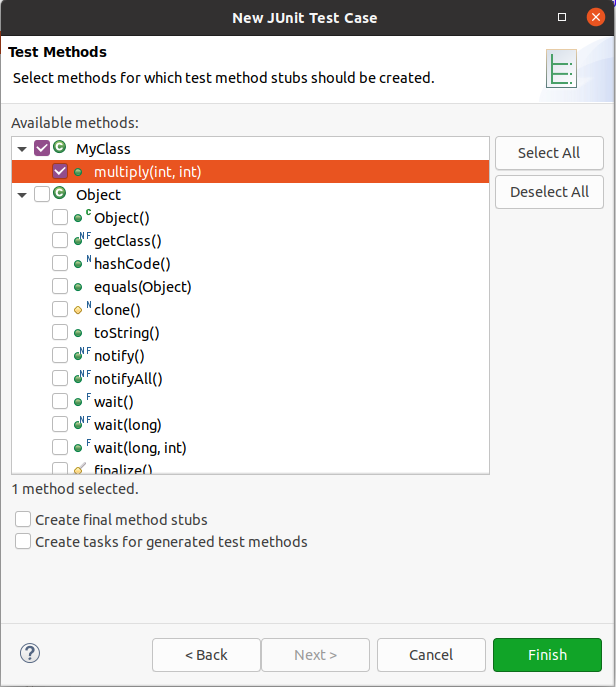


|  |  |
| --- | --- |
|  | Alternatively you can right-click on your new class in the :\_Project Explorer\_ or Package Explorer view and select **New**  **Other**  **Java**  **JUnit Test Case**. |

In the following wizard ensure that the New JUnit Jupiter test flag is selected. The source folder should select the test directory.



Press the Next button and select the methods that you want to test.



Create a test with the following code.

**package** **com.vogella.junit.first;**

**import** **static** org**.**junit**.**jupiter**.**api**.**Assertions**.**assertAll**;**

**import** **static** org**.**junit**.**jupiter**.**api**.**Assertions**.**assertEquals**;**

**import** **static** org**.**junit**.**jupiter**.**api**.**Assertions**.**assertThrows**;**

**import** **org.junit.jupiter.api.BeforeEach;**

**import** **org.junit.jupiter.api.Test;**

**class** **MyClassTest** **{**

**@Test**

**void** testExceptionIsThrown**()** **{**

**MyClass** tester **=** **new** **MyClass();**

assertThrows**(IllegalArgumentException.**class**,** **()** **->** tester**.**multiply**(1000,** **5));**

**}**

**@Test**

**void** testMultiply**()** **{**

**MyClass** tester **=** **new** **MyClass();**

assertEquals**(50,** tester**.**multiply**(10,** **5),** **"10 x 5 must be 50");**

**}**

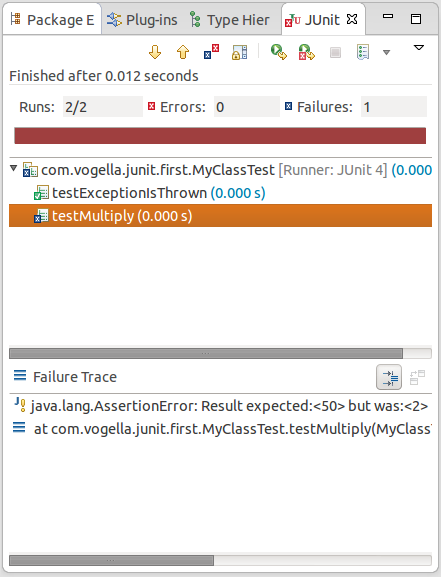
**}**

### [5.6. Run your test in Eclipse](https://www.vogella.com/tutorials/JUnit/article.html#run-your-test-in-eclipse)

Right-click on your new test class and select **Run-As**  **JUnit Test**.

Run JUnit test in Eclipse

The result of the tests are displayed in the JUnit view. In our example one test should be successful and one test should show an error. This error is indicated by a red bar.



You discovered a bug in the tested code!

### [5.7. Fix the bug and re-run your tests](https://www.vogella.com/tutorials/JUnit/article.html#fix-the-bug-and-re-run-your-tests)

The test is failing, because our multiplier class is currently not working correctly. It does a division instead of multiplication. Fix the bug and re-run the test to get a green bar.

Solution

### [5.8. Review](https://www.vogella.com/tutorials/JUnit/article.html#review)

After a few minutes you should have created a new project, a new class and a new unit test. Congratulations! If you feel like it, lets improve the tests a bit and write one grouped test.

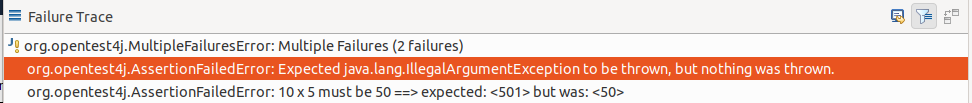
### [5.9. Simplify your test code with @Before each](https://www.vogella.com/tutorials/JUnit/article.html#simplify-your-test-code-with-before-each)

The initialization of MyClass happens in every test, move the initialization to a @BeforeEach method.

Solution

### [5.10. Define a group check with assertAll](https://www.vogella.com/tutorials/JUnit/article.html#define-a-group-check-with-assertall)

Define a new test method which checks both condition at the same time with assertAll statement. Change the condition to make both tests fail, run the test and ensure that both are executed.



Solution

Afterwards adjust the test so that both are successfully executed.

## [6. Exercise: Writing a JUnit 5 test with Gradle and Eclipse in 5 mins](https://www.vogella.com/tutorials/JUnit/article.html#junitgradleeclipse)

In this exercise you learn you to write a JUnit5 test using the Gradle build system and the Eclipse IDE.

### [6.1. Project creation](https://www.vogella.com/tutorials/JUnit/article.html#project-creation-2)

Create a new Gradle project with the following setting:

* Name: com.vogella.junit.first

See [Create a Grade project with Eclipse](https://www.vogella.com/tutorials/EclipseGradle/article.html#how-to-create-a-gradle-project-for-java-with-eclipse) to learn how to create a Gradle project with Eclipse.

The wizard should also have create the package com.vogella.junit.first in the src/main/java and src/main/test folder. Remove the generated classes from it.

### [6.2. Update build.gradle file to use JUnit5](https://www.vogella.com/tutorials/JUnit/article.html#update-build-gradle-file-to-use-junit5)

To use JUnit 5 with the Gradle build system, ensure you use at least Gradle 6.0 to avoid already fixed issues.

Modify your build.gradle file to contain at least the following entries. Your build file may contain more dependencies.

plugins **{**

id **'java'**

**}**

**repositories** **{**

mavenCentral**()**

**}**

**dependencies** **{**

testImplementation **'org.junit.jupiter:junit-jupiter:5.7.2'**

**}**

test **{**

useJUnitPlatform**()**

### [6.3. Create a Java class](https://www.vogella.com/tutorials/JUnit/article.html#create-a-java-class-2)

In the src folder, create the following class in the com.vogella.junit.first package.

**package** **com.vogella.junit.first;**

**public** **class** **MyClass** **{**

*// the following is just an example*

**public** **int** multiply**(int** x**,** **int** y**)** **{**

**if** **(**x **>** **999)** **{**

**throw** **new** IllegalArgumentException**("X should be less than 1000");**

**}**

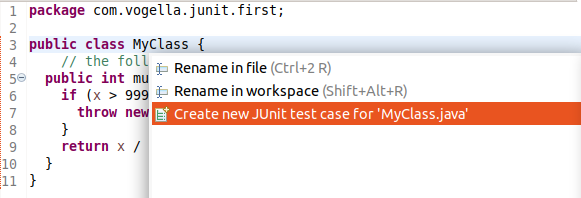
**return** x **/** y**;**

**}**

**}**

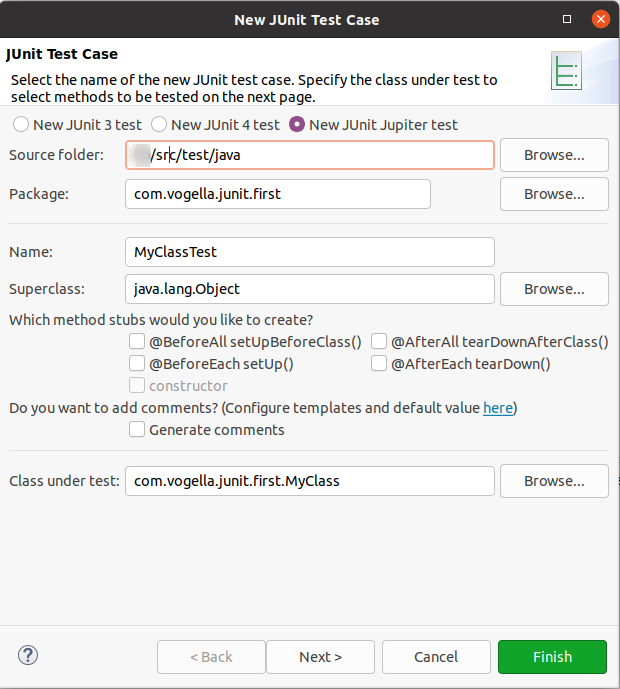
### [6.4. Create a JUnit test](https://www.vogella.com/tutorials/JUnit/article.html#create-a-junit-test-2)

Position the cursor on the MyClass in the Java editor and press Ctrl+1. Select that you want to create a new JUnit test from the list.

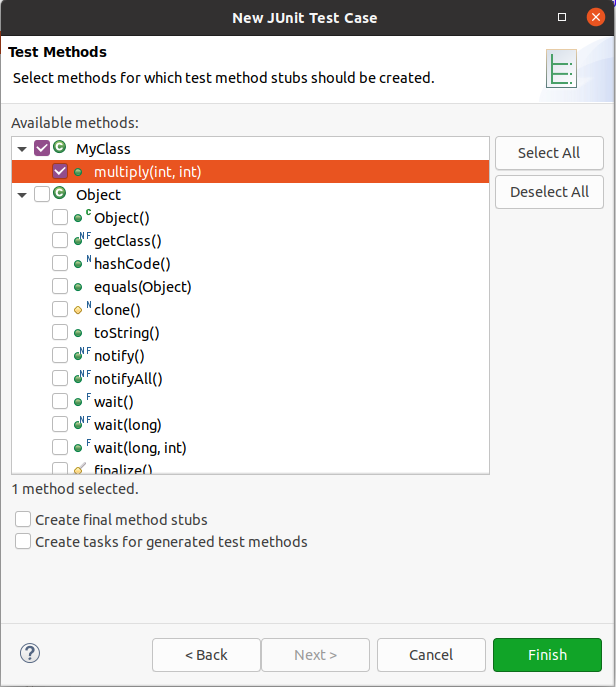


|  |  |
| --- | --- |
|  | Alternatively you can right-click on your new class in the :\_Project Explorer\_ or Package Explorer view and select **New**  **Other**  **Java**  **JUnit Test Case**. |

In the following wizard ensure that the New JUnit Jupiter test flag is selected. The source folder should select the test directory.



Press the Next button and select the methods that you want to test.



Create a test with the following code.

**package** **com.vogella.junit.first;**

**import** **static** org**.**junit**.**jupiter**.**api**.**Assertions**.**assertAll**;**

**import** **static** org**.**junit**.**jupiter**.**api**.**Assertions**.**assertEquals**;**

**import** **static** org**.**junit**.**jupiter**.**api**.**Assertions**.**assertThrows**;**

**import** **org.junit.jupiter.api.BeforeEach;**

**import** **org.junit.jupiter.api.Test;**

**class** **MyClassTest** **{**

**@Test**

**void** testExceptionIsThrown**()** **{**

**MyClass** tester **=** **new** **MyClass();**

assertThrows**(IllegalArgumentException.**class**,** **()** **->** tester**.**multiply**(1000,** **5));**

**}**

**@Test**

**void** testMultiply**()** **{**

**MyClass** tester **=** **new** **MyClass();**

assertEquals**(50,** tester**.**multiply**(10,** **5),** **"10 x 5 must be 50");**

**}**

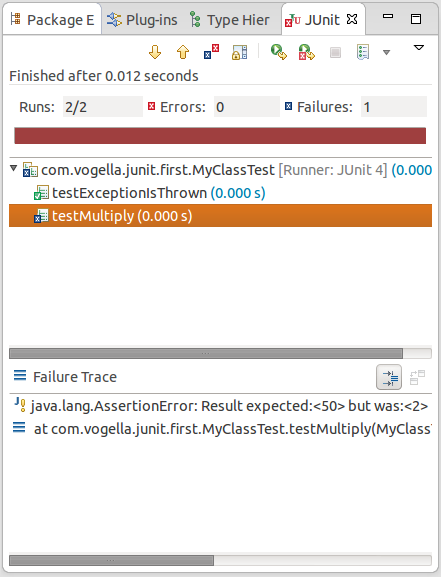
**}**

### [6.5. Run your test in Eclipse](https://www.vogella.com/tutorials/JUnit/article.html#run-your-test-in-eclipse-2)

Right-click on your new test class and select **Run-As**  **JUnit Test**.

Run JUnit test in Eclipse

The result of the tests are displayed in the JUnit view. In our example one test should be successful and one test should show an error. This error is indicated by a red bar.



You discovered a bug in the tested code!

### [6.6. Fix the bug and re-run your tests](https://www.vogella.com/tutorials/JUnit/article.html#fix-the-bug-and-re-run-your-tests-2)

The test is failing, because our multiplier class is currently not working correctly. It does a division instead of multiplication. Fix the bug and re-run the test to get a green bar.

Solution

### [6.7. Review](https://www.vogella.com/tutorials/JUnit/article.html#review-2)

After a few minutes you should have created a new project, a new class and a new unit test. Congratulations! If you feel like it, lets improve the tests a bit and write one grouped test.

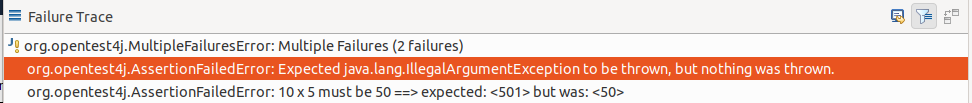
### [6.8. Simplify your test code with @Before each](https://www.vogella.com/tutorials/JUnit/article.html#simplify-your-test-code-with-before-each-2)

The initialization of MyClass happens in every test, move the initialization to a @BeforeEach method.

Solution

### [6.9. Define a group check with assertAll](https://www.vogella.com/tutorials/JUnit/article.html#define-a-group-check-with-assertall-2)

Define a new test method which checks both condition at the same time with assertAll statement. Change the condition to make both tests fail, run the test and ensure that both are executed.



Solution

Afterwards adjust the test so that both are successfully executed.

## [7. Exercise: Writing JUnit5 unit tests](https://www.vogella.com/tutorials/JUnit/article.html#exercise-writing-junit5-unit-tests)

In this exercise, you develop some JUnit 5 tests for a given data model. You already learned how to create projects with Maven or Gradle and how to add Junit5 to them.

To review this information see:

* [Create a Grade project with Eclipse](https://www.vogella.com/tutorials/EclipseGradle/article.html#how-to-create-a-gradle-project-for-java-with-eclipse)
* [Adding JUnit to an project](https://www.vogella.com/tutorials/JUnit/article.html#junitsetup)

The following description assumes that you are familiar with these steps and will not repeat them.

### [7.1. Create project and add JUnit5 dependencies](https://www.vogella.com/tutorials/JUnit/article.html#create-project-and-add-junit5-dependencies)

Create a new project called com.vogella.unittest either with Maven or with Gradle and update their settings to use JUnit5.

### [7.2. Create the data model used for testing](https://www.vogella.com/tutorials/JUnit/article.html#create-the-data-model-used-for-testing)

Create the com.vogella.unittest.model package and copy and paste the following classes on it.

**package** **com.vogella.unittest.model;**

**import** **java.util.Date;**

**public** **class** **Movie** **{**

**private** **String** title**;**

**private** **Date** releaseDate**;**

**@SuppressWarnings("unused")**

**private** **String** duration**;**

**public** Movie**(String** title**,** **Date** releaseDate**,** **String** duration**)** **{**

**super();**

**this.**title **=** title**;**

**this.**releaseDate **=** releaseDate**;**

**this.**duration **=** duration**;**

**}**

**public** **String** getTitle**()** **{**

**return** title**;**

**}**

**public** **Date** getReleaseDate**()** **{**

**return** releaseDate**;**

**}**

**@Override**

**public** **int** hashCode**()** **{**

**final** **int** prime **=** **31;**

**int** result **=** **1;**

result **=** prime **\*** result **+** **((**releaseDate **==** **null)** **?** **0** **:** releaseDate**.**hashCode**());**

result **=** prime **\*** result **+** **((**title **==** **null)** **?** **0** **:** title**.**hashCode**());**

**return** result**;**

**}**

**@Override**

**public** **boolean** equals**(Object** obj**)** **{**

**if** **(this** **==** obj**)** **return** **true;**

**if** **(**obj **==** **null)** **return** **false;**

**if** **(**getClass**()** **!=** obj**.**getClass**())** **return** **false;**

**Movie** other **=** **(Movie)** obj**;**

**if** **(**releaseDate **==** **null)** **{**

**if** **(**other**.**releaseDate **!=** **null)** **return** **false;**

**}** **else** **if** **(!**releaseDate**.**equals**(**other**.**releaseDate**))** **return** **false;**

**if** **(**title **==** **null)** **{**

**if** **(**other**.**title **!=** **null)** **return** **false;**

**}** **else** **if** **(!**title**.**equals**(**other**.**title**))** **return** **false;**

**return** **true;**

**}**

**}**

**package** **com.vogella.unittest.model;**

**public** **enum** **Alignment** **{**

**SUPER\_EVIL,** **EVIL,** **NEUTRAL,** **GOOD,** **SUPER\_GOOD;**

**}**

**package** **com.vogella.unittest.model;**

**import** **java.lang.annotation.ElementType;**

**import** **java.lang.annotation.Retention;**

**import** **java.lang.annotation.RetentionPolicy;**

**import** **java.lang.annotation.Target;**

**@Target(ElementType.**TYPE**)**

**@Retention(RetentionPolicy.**RUNTIME**)**

**public** **@interface** **Magical** **{**

**}**

**package** **com.vogella.unittest.model;**

*/\*\**

*\* Race in Tolkien's Lord of the Rings.*

*\**

*\* @author Florent Biville*

*\*/*

**public** **enum** **Race** **{**

**HOBBIT("Hobbit",** **false,** **Alignment.**GOOD**),** **MAIA("Maia",** **true,** **Alignment.**GOOD**),** **MAN("Man",** **false,** **Alignment.**NEUTRAL**),** **ELF("Elf",** **true,** **Alignment.**GOOD**),** **DWARF("Dwarf",** **false,** **Alignment.**GOOD**),** **ORC("Orc",** **false,** **Alignment.**EVIL**);**

**private** **final** **String** name**;**

**private** **final** **boolean** immortal**;**

**private** **Alignment** alignment**;**

**Race(String** name**,** **boolean** immortal**,** **Alignment** alignment**)** **{**

**this.**name **=** name**;**

**this.**immortal **=** immortal**;**

**this.**alignment **=** alignment**;**

**}**

**public** **String** getName**()** **{**

**return** name**;**

**}**

**public** **boolean** isImmortal**()** **{**

**return** immortal**;**

**}**

**public** **Alignment** getAlignment**()** **{**

**return** alignment**;**

**}**

**@Override**

**public** **String** toString**()** **{**

**return** **"Race [name="** **+** name **+** **", immortal="** **+** immortal **+** **"]";**

**}**

**}**

**package** **com.vogella.unittest.model;**

**@Magical**

**public** **enum** **Ring** **{**

oneRing**,** vilya**,** nenya**,** narya**,** dwarfRing**,** manRing**;**

**}**

**package** **com.vogella.unittest.model;**

**public** **class** **TolkienCharacter** **{**

*// public to test extract on field*

**public** **int** age**;**

**private** **String** name**;**

**private** **Race** race**;**

*// not accessible field to test that field by field comparison does not use it*

**@SuppressWarnings("unused")**

**private** **long** notAccessibleField **=** **System.**currentTimeMillis**();**

**public** TolkienCharacter**(String** name**,** **int** age**,** **Race** race**)** **{**

**this.**name **=** name**;**

**this.**age **=** age**;**

**this.**race **=** race**;**

**}**

**public** **Race** getRace**()** **{**

**return** race**;**

**}**

**public** **String** getName**()** **{**

**return** name**;**

**}**

**public** **void** setName**(String** name**)** **{**

**this.**name **=** name**;**

**}**

**public** **void** setAge**(int** age**)** **{**

**if** **(**age**<0)** **{**

**throw** **new** IllegalArgumentException**("Age is not allowed to be smaller than zero");**

**}**

**this.**age **=** age**;**

**}**

**@Override**

**public** **int** hashCode**()** **{**

**final** **int** prime **=** **31;**

**int** result **=** **1;**

result **=** prime **\*** result **+** age**;**

result **=** prime **\*** result **+** **((**name **==** **null)** **?** **0** **:** name**.**hashCode**());**

result **=** prime **\*** result **+** **((**race **==** **null)** **?** **0** **:** race**.**hashCode**());**

**return** result**;**

**}**

**@Override**

**public** **boolean** equals**(Object** obj**)** **{**

**if** **(this** **==** obj**)** **return** **true;**

**if** **(**obj **==** **null)** **return** **false;**

**if** **(**getClass**()** **!=** obj**.**getClass**())** **return** **false;**

**TolkienCharacter** other **=** **(TolkienCharacter)** obj**;**

**if** **(**age **!=** other**.**age**)** **return** **false;**

**if** **(**name **==** **null)** **{**

**if** **(**other**.**name **!=** **null)** **return** **false;**

**}** **else** **if** **(!**name**.**equals**(**other**.**name**))** **return** **false;**

**if** **(**race **==** **null)** **{**

**if** **(**other**.**race **!=** **null)** **return** **false;**

**}** **else** **if** **(!**race**.**equals**(**other**.**race**))** **return** **false;**

**return** **true;**

**}**

**@Override**

**public** **String** toString**()** **{**

**return** name **+** **" "** **+** age **+** **" years old "** **+** race**.**getName**();**

**}**

**}**

Create the com.vogella.unittest.services package and copy and paste the following classes on it.

**package** **com.vogella.unittest.services;**

**import** **static** com**.**vogella**.**unittest**.**model**.**Race**.**DWARF**;**

**import** **static** com**.**vogella**.**unittest**.**model**.**Race**.**ELF**;**

**import** **static** com**.**vogella**.**unittest**.**model**.**Race**.**HOBBIT**;**

**import** **static** com**.**vogella**.**unittest**.**model**.**Race**.**MAIA**;**

**import** **static** com**.**vogella**.**unittest**.**model**.**Race**.**MAN**;**

**import** **static** com**.**vogella**.**unittest**.**model**.**Race**.**ORC**;**

**import** **java.util.ArrayList;**

**import** **java.util.Date;**

**import** **java.util.HashMap;**

**import** **java.util.List;**

**import** **java.util.Map;**

**import** **com.vogella.unittest.model.Movie;**

**import** **com.vogella.unittest.model.Ring;**

**import** **com.vogella.unittest.model.TolkienCharacter;**

*/\*\**

*\* Init data for unit test*

*\*/*

**public** **class** **DataService** **{**

**static** **final** **String** **ERROR\_MESSAGE\_EXAMPLE\_FOR\_ASSERTION** **=** **"{} assertion : {}\n";**

*// Some of the Lord of the Rings characters :*

**final** **TolkienCharacter** frodo **=** **new** **TolkienCharacter("Frodo",** **33,** **HOBBIT);**

**final** **TolkienCharacter** sam **=** **new** **TolkienCharacter("Sam",** **38,** **HOBBIT);**

**final** **TolkienCharacter** merry **=** **new** **TolkienCharacter("Merry",** **36,** **HOBBIT);**

**final** **TolkienCharacter** pippin **=** **new** **TolkienCharacter("Pippin",** **28,** **HOBBIT);**

**final** **TolkienCharacter** gandalf **=** **new** **TolkienCharacter("Gandalf",** **2020,** **MAIA);**

**final** **TolkienCharacter** gimli **=** **new** **TolkienCharacter("Gimli",** **139,** **DWARF);**

**final** **TolkienCharacter** legolas **=** **new** **TolkienCharacter("Legolas",** **1000,** **ELF);**

**final** **TolkienCharacter** aragorn **=** **new** **TolkienCharacter("Aragorn",** **87,** **MAN);**

**final** **TolkienCharacter** boromir **=** **new** **TolkienCharacter("Boromir",** **37,** **MAN);**

**final** **TolkienCharacter** sauron **=** **new** **TolkienCharacter("Sauron",** **50000,** **MAIA);**

**final** **TolkienCharacter** galadriel **=** **new** **TolkienCharacter("Galadriel",** **3000,** **ELF);**

**final** **TolkienCharacter** elrond **=** **new** **TolkienCharacter("Elrond",** **3000,** **ELF);**

**final** **TolkienCharacter** guruk **=** **new** **TolkienCharacter("Guruk",** **20,** **ORC);**

**final** **Movie** theFellowshipOfTheRing **=** **new** **Movie("the fellowship of the Ring",** **new** **Date(),** **"178 min");**

**final** **Movie** theTwoTowers **=** **new** **Movie("the two Towers",** **new** **Date(),** **"179 min");**

**final** **Movie** theReturnOfTheKing **=** **new** **Movie("the Return of the King",** **new** **Date(),** **"201 min");**

**public** **List<TolkienCharacter>** getFellowship**()** **{**

**final** **List<TolkienCharacter>** fellowshipOfTheRing **=** **new** **ArrayList<>();**

*// let's do some team building :)*

fellowshipOfTheRing**.**add**(**frodo**);**

fellowshipOfTheRing**.**add**(**sam**);**

fellowshipOfTheRing**.**add**(**merry**);**

fellowshipOfTheRing**.**add**(**pippin**);**

fellowshipOfTheRing**.**add**(**gandalf**);**

fellowshipOfTheRing**.**add**(**legolas**);**

fellowshipOfTheRing**.**add**(**gimli**);**

fellowshipOfTheRing**.**add**(**aragorn**);**

fellowshipOfTheRing**.**add**(**boromir**);**

**return** fellowshipOfTheRing**;**

**}**

**public** **List<TolkienCharacter>** getOrcsWithHobbitPrisoners**()** **{**

**final** **List<TolkienCharacter>** orcsWithHobbitPrisoners **=** **new** **ArrayList<TolkienCharacter>();**

orcsWithHobbitPrisoners**.**add**(**guruk**);**

orcsWithHobbitPrisoners**.**add**(**merry**);**

orcsWithHobbitPrisoners**.**add**(**pippin**);**

**return** orcsWithHobbitPrisoners**;**

**}**

**public** **TolkienCharacter** getFellowshipCharacter**(String** name**)** **{**

**List<TolkienCharacter>** list **=** getFellowship**();**

**return** list**.**stream**().**filter**(**s**->** s**.**equals**(**name**)).**findFirst**().**get**();**

**}**

**public** **Map<Ring,** **TolkienCharacter>** getRingBearers**()** **{**

**Map<Ring,** **TolkienCharacter>** ringBearers **=** **new** **HashMap<>();**

*// ring bearers*

ringBearers**.**put**(Ring.**nenya**,** galadriel**);**

ringBearers**.**put**(Ring.**narya**,** gandalf**);**

ringBearers**.**put**(Ring.**vilya**,** elrond**);**

ringBearers**.**put**(Ring.**oneRing**,** frodo**);**

**return** ringBearers**;**

**}**

**}**

### [7.3. Write tests for the model and the services](https://www.vogella.com/tutorials/JUnit/article.html#write-tests-for-the-model-and-the-services)

Create the following test class.

**package** **com.vogella.unittest.services;**

**import** **static** com**.**vogella**.**unittest**.**model**.**Race**.**HOBBIT**;**

**import** **static** org**.**junit**.**jupiter**.**api**.**Assertions**.**assertNotNull**;**

**import** **static** org**.**junit**.**jupiter**.**api**.**Assertions**.**fail**;**

**import** **java.util.List;**

**import** **org.junit.jupiter.api.DisplayName;**

**import** **org.junit.jupiter.api.Tag;**

**import** **org.junit.jupiter.api.Test;**

**import** **com.vogella.unittest.model.TolkienCharacter;**

**import** **com.vogella.unittest.services.DataService;**

**class** **DataServiceTest** **{**

*// TODO initialize before each test*

**DataService** dataService**;**

**@Test**

**void** ensureThatInitializationOfTolkeinCharactorsWorks**()** **{**

**TolkienCharacter** frodo **=** **new** **TolkienCharacter("Frodo",** **33,** **HOBBIT);**

*// TODO check that age is 33*

*// TODO check that name is "Frodo"*

*// TODO check that name is not "Frodon"*

fail**("not yet implemented");**

**}**

**@Test**

**void** ensureThatEqualsWorksForCharaters**()** **{**

**Object** jake **=** **new** **TolkienCharacter("Jake",** **43,** **HOBBIT);**

**Object** sameJake **=** jake**;**

**Object** jakeClone **=** **new** **TolkienCharacter("Jake",** **12,** **HOBBIT);**

*// TODO check that:*

*// jake is equal to sameJake*

*// jake is not equal to jakeClone*

fail**("not yet implemented");**

**}**

**@Test**

**void** checkInheritance**()** **{**

**TolkienCharacter** tolkienCharacter **=** dataService**.**getFellowship**().**get**(0);**

*// TODO check that tolkienCharacter.getClass is not a movie class*

fail**("not yet implemented");**

**}**

**@Test**

**void** ensureFellowShipCharacterAccessByNameReturnsNullForUnknownCharacter**()** **{**

*// TODO imlement a check that dataService.getFellowshipCharacter returns null for an*

*// unknow felllow, e.g. "Lars"*

fail**("not yet implemented");**

**}**

**@Test**

**void** ensureFellowShipCharacterAccessByNameWorksGivenCorrectNameIsGiven**()** **{**

*// TODO imlement a check that dataService.getFellowshipCharacter returns a fellow for an*

*// existing felllow, e.g. "Frodo"*

fail**("not yet implemented");**

**}**

**@Test**

**void** ensureThatFrodoAndGandalfArePartOfTheFellowsip**()** **{**

**List<TolkienCharacter>** fellowship **=** dataService**.**getFellowship**();**

*// TODO check that Frodo and Gandalf are part of the fellowship*

fail**("not yet implemented");**

**}**

**@Test**

**void** ensureThatOneRingBearerIsPartOfTheFellowship**()** **{**

**List<TolkienCharacter>** fellowship **=** dataService**.**getFellowship**();**

*// TODO test that at least one ring bearer is part of the fellowship*

fail**("not yet implemented");**

**}**

*// TODO Use @RepeatedTest(int) to execute this test 1000 times*

**@Test**

**@Tag("slow")**

**@DisplayName("Minimal stress testing: run this test 1000 times to ")**

**void** ensureThatWeCanRetrieveFellowshipMultipleTimes**()** **{**

dataService **=** **new** **DataService();**

assertNotNull**(**dataService**.**getFellowship**());**

fail**("this should run 1000 times");**

**}**

**@Test**

**void** ensureOrdering**()** **{**

**List<TolkienCharacter>** fellowship **=** dataService**.**getFellowship**();**

*// ensure that the order of the fellowship is:*

*// frodo, sam, merry,pippin, gandalf,legolas,gimli,aragorn,boromir*

fail**("not yet implemented");**

**}**

**@Test**

**void** ensureAge**()** **{**

**List<TolkienCharacter>** fellowship **=** dataService**.**getFellowship**();**

*// TODO test ensure that all hobbits and men are younger than 100 years*

*// TODO also ensure that the elfs, dwars the maia are all older than 100 years*

fail**("not yet implemented");**

*// HINT fellowship.stream might be useful here*

**}**

**@Test**

**void** ensureThatFellowsStayASmallGroup**()** **{**

**List<TolkienCharacter>** fellowship **=** dataService**.**getFellowship**();**

*// TODO Write a test to get the 20 element from the fellowship throws an*

*// IndexOutOfBoundsException*

fail**("not yet implemented");**

**}**

**}**

Solve the TODO and ensure that all tests can be successfully executed from your IDE. You may find issues in the DataService with these tests, fix them if you encounter them.

Solution

### [7.4. Verify on command line](https://www.vogella.com/tutorials/JUnit/article.html#verify-on-command-line)

Verify that your code compiles and your test are running via the command line with:

* mvn clean verify in case you are using Maven
* ./gradlew test in case you are using Gradle

### [7.5. Add a long running method to your data service](https://www.vogella.com/tutorials/JUnit/article.html#add-a-long-running-method-to-your-data-service)

Add a fake update method to your DataService which takes a long time to update the data and returns true on success.

**public** **boolean** update**()** **{**

**try** **{**

**Thread.**sleep**(2000);**

**}** **catch** **(InterruptedException** e**)** **{**

e**.**printStackTrace**();**

**}**

**return** **true;**

**}**

### [7.6. Develop a test to constrain the execution time of the long running method](https://www.vogella.com/tutorials/JUnit/article.html#develop-a-test-to-constrain-the-execution-time-of-the-long-running-method)

Create a new test method in your DataServiceTest. Use the assertTimeout assert statement to ensure that this test does not run longer than 3 seconds.

Solution

## [8. Exercise: Develop unit tests for a regular expression utility method for email verification](https://www.vogella.com/tutorials/JUnit/article.html#exercise-develop-unit-tests-for-a-regular-expression-utility-method-for-email-verification)

### [8.1. Create the data model used for testing](https://www.vogella.com/tutorials/JUnit/article.html#create-the-data-model-used-for-testing-2)

Create the com.vogella.unittest.email package and copy and paste the following classes on it.

**package** **com.vogella.unittest.email;**

**import** **java.util.regex.Pattern;**

**public** **class** **EmailValidator** **{**

*/\*\**

*\* Email validation pattern.*

*\*/*

**public** **static** **final** **Pattern** **EMAIL\_PATTERN** **=** **Pattern.**compile**(**

**"[a-zA-Z0-9\\+\\.\\\_\\%\\-\\+]{1,256}"** **+**

**"\\@"** **+**

**"[a-zA-Z0-9][a-zA-Z0-9\\-]{0,64}"** **+**

**"("** **+**

**"\\."** **+**

**"[a-zA-Z0-9][a-zA-Z0-9\\-]{0,25}"** **+**

**")+"**

**);**

**private** **boolean** mIsValid **=** **false;**

**public** **boolean** isValid**()** **{**

**return** mIsValid**;**

**}**

*/\*\**

*\* Validates if the given input is a valid email address.*

*\**

*\* @param emailPattern The {@link Pattern} used to validate the given email.*

*\* @param email The email to validate.*

*\* @return {@code true} if the input is a valid email. {@code false} otherwise.*

*\*/*

**public** **static** **boolean** isValidEmail**(CharSequence** email**)** **{**

**return** email **!=** **null** **&&** **EMAIL\_PATTERN.**matcher**(**email**).**matches**();**

**}**

**}**

### [8.2. Write tests for the model and the services](https://www.vogella.com/tutorials/JUnit/article.html#write-tests-for-the-model-and-the-services-2)

Create the following test class.

**package** **com.vogella.unittest.email;**

**import** **static** org**.**junit**.**jupiter**.**api**.**Assertions**.**assertFalse**;**

**import** **static** org**.**junit**.**jupiter**.**api**.**Assertions**.**assertTrue**;**

**import** **static** org**.**junit**.**jupiter**.**api**.**Assertions**.**fail**;**

**import** **org.junit.jupiter.api.DisplayName;**

**import** **org.junit.jupiter.api.Test;**

**class** **EmailValidatorTest** **{**

*// TODO Write test for EmailValidator*

*// The names of the methods should give you a pointer what to test for*

**@Test**

**public** **void** ensureThatEmailValidatorReturnsTrueForValidEmail**()** **{**

assertTrue**(EmailValidator.**isValidEmail**("lars.vogel@gmail.com"));**

**}**

**@Test**

**@DisplayName("Ensure that the usage of a subdomain is still valid, see https://en.wikipedia.org/wiki/Subdomain")**

**public** **void** emailValidator\_CorrectEmailSubDomain\_ReturnsTrue**()** **{**

fail**("Fixme");**

**}**

**@Test**

**@DisplayName("Ensure that a missiong top level domain returns false")**

**public** **void** emailValidator\_InvalidEmailNoTld\_ReturnsFalse**()** **{**

fail**("Fixme");**

**}**

**@Test**

**public** **void** emailValidator\_InvalidEmailDoubleDot\_ReturnsFalse**()** **{**

fail**("Fixme");**

**}**

**@Test**

**public** **void** emailValidator\_InvalidEmailNoUsername\_ReturnsFalse**()** **{**

fail**("Fixme");**

**}**

**@Test**

**public** **void** emailValidator\_EmptyString\_ReturnsFalse**()** **{**

fail**("Fixme");**

**}**

**@Test**

**public** **void** emailValidator\_NullEmail\_ReturnsFalse**()** **{**

fail**("Fixme");**

**}**

**}**

Fix all the failing test, unfortunately the test specification is not very good. Try to write reasonable tests which fit the method name.

### [8.3. Verify](https://www.vogella.com/tutorials/JUnit/article.html#verify)

Run your new test via the IDE. Verify that your code compiles and your test are running via the command line.

### [8.4. Solution](https://www.vogella.com/tutorials/JUnit/article.html#solution)

The following listing contains a possible implementation of the test.

Solution

## [9. Exercise: Testing exceptions and conditional enablement](https://www.vogella.com/tutorials/JUnit/article.html#exercise-testing-exceptions-and-conditional-enablement)

### [9.1. Write tests checking for exceptions](https://www.vogella.com/tutorials/JUnit/article.html#write-tests-checking-for-exceptions)

We also want to check that exceptions with the correct error messages are thrown, if we call the class under test with incorrect data.

Create the following test class.

**package** **com.vogella.unittest.services;**

**import** **static** com**.**vogella**.**unittest**.**model**.**Race**.**HOBBIT**;**

**import** **static** org**.**junit**.**jupiter**.**api**.**Assertions**.**assertEquals**;**

**import** **static** org**.**junit**.**jupiter**.**api**.**Assertions**.**assertThrows**;**

**import** **static** org**.**junit**.**jupiter**.**api**.**Assertions**.**fail**;**

**import** **org.junit.jupiter.api.Disabled;**

**import** **org.junit.jupiter.api.DisplayName;**

**import** **org.junit.jupiter.api.Test;**

**import** **com.vogella.unittest.model.TolkienCharacter;**

**public** **class** **DataModelAssertThrowsTest** **{**

**@Test**

**@DisplayName("Ensure that access to the fellowship throws exception outside the valid range")**

**void** exceptionTesting**()** **{**

**DataService** dataService **=** **new** **DataService();**

**Throwable** exception **=** assertThrows**(IndexOutOfBoundsException.**class**,** **()** **->** dataService**.**getFellowship**().**get**(20));**

assertEquals**("Index 20 out of bounds for length 9",** exception**.**getMessage**());**

**}**

**@Test**

**@Disabled("Please fix and enable")**

**public** **void** ensureThatAgeMustBeLargerThanZeroViaSetter**()** **{**

**TolkienCharacter** frodo **=** **new** **TolkienCharacter("Frodo",** **33,** **HOBBIT);**

*// use assertThrows() rule to check that the message is:*

*// Age is not allowed to be smaller than zero*

frodo**.**setAge**(-1);**

**}**

**@Test**

**@Disabled("Please fix and enable")**

**public** **void** testThatAgeMustBeLargerThanZeroViaConstructor**()** **{**

*// use assertThrows() rule to check that an IllegalArgumentException exception is thrown and*

*// that the message is:*

*// "Age is not allowed to be smaller than zero"*

**TolkienCharacter** frodo **=** **new** **TolkienCharacter("Frodo",** **-1,** **HOBBIT);**

**}**

**}**

Fix the disabled tests and enable them. The name should give a good indication what you have to do test here.

You may discover that the data model does not behave a expected by the test, fix them in this case.

### [9.2. Verify](https://www.vogella.com/tutorials/JUnit/article.html#verify-2)

Run your update test via the IDE. Verify that your code compiles and your test are running via the command line with the mvn clean verify.

### [9.3. Solution](https://www.vogella.com/tutorials/JUnit/article.html#solution-2)

Solution

### [9.4. Enable test only on certain platforms](https://www.vogella.com/tutorials/JUnit/article.html#enable-test-only-on-certain-platforms)

Write this test and adjust it so that is only runs on the operating system your are using.

**package** **com.vogella.unittest.platform;**

**import** **static** org**.**junit**.**jupiter**.**api**.**Assertions**.**assertTrue**;**

**import** **org.junit.jupiter.api.Assumptions;**

**import** **org.junit.jupiter.api.Test;**

**class** **LinuxTests** **{**

**@Test**

**void** testName**()** **throws** **Exception** **{**

*// only run on Linux*

**Assumptions.**assumeTrue**(System.**getProperty**("os.name").**contains**("Linux"));**

assertTrue**(true);**

**}**

**}**

## [10. Exercise: Writing nested tests to group tests for display](https://www.vogella.com/tutorials/JUnit/article.html#exercise-writing-nested-tests-to-group-tests-for-display)

### [10.1. Write tests](https://www.vogella.com/tutorials/JUnit/article.html#write-tests)

Create the following test.

**package** **com.vogella.unittest.nested;**

**import** **static** org**.**junit**.**jupiter**.**api**.**Assertions**.**assertEquals**;**

**import** **java.util.Arrays;**

**import** **java.util.List;**

**import** **org.junit.jupiter.api.BeforeEach;**

**import** **org.junit.jupiter.api.DisplayName;**

**import** **org.junit.jupiter.api.Nested;**

**import** **org.junit.jupiter.api.Test;**

**class** **UsingNestedTests** **{**

**private** **List<String>** list**;**

**@BeforeEach**

**void** setup**()** **{**

list **=** **Arrays.**asList**("JUnit 5",** **"Mockito");**

**}**

**@Test**

**void** listTests**()** **{**

assertEquals**(2,** list**.**size**());**

**}**

*// TODO define inner class with @Nestled*

*// write one tests named checkFirstElement() to check that the first list element is "JUnit 4"*

*// write one tests named checkSecondElement() to check that the first list element is "JUnit 4"*

**@DisplayName("Grouped tests for checking members")**

**@Nested**

**class** **CheckMembers** **{**

**@Test**

**void** checkFirstElement**()** **{**

assertEquals**(("JUnit 5"),** list**.**get**(0));**

**}**

**@Test**

**void** checkSecondElement**()** **{**

assertEquals**(("Mockito"),** list**.**get**(1));**

**}**

**}**

**}**

### [10.2. Solution](https://www.vogella.com/tutorials/JUnit/article.html#solution-3)

The following listing contains a possible implementation of the test.

Solution

### [10.3. Run tests](https://www.vogella.com/tutorials/JUnit/article.html#run-tests)

Run the test from your IDE and review how the grouped tests are displayed.

## [11. Exercise: Testing multiple parameter](https://www.vogella.com/tutorials/JUnit/article.html#exercise-testing-multiple-parameter)

### [11.1. Create class for testing](https://www.vogella.com/tutorials/JUnit/article.html#create-class-for-testing)

Create the com.vogella.unittest.converter package and copy and paste the following class on it.

**package** **com.vogella.unittest.converter;**

**public** **class** **ConverterUtil** **{**

*// converts to celsius*

**public** **static** **float** convertFahrenheitToCelsius**(float** fahrenheit**)** **{**

**return** **((**fahrenheit **-** **32)** **\*** **5** **/** **9);**

**}**

*// converts to fahrenheit*

**public** **static** **float** convertCelsiusToFahrenheit**(float** celsius**)** **{**

**return** **((**celsius **\*** **9)** **/** **5)** **+** **32;**

**}**

**}**

### [11.2. Write a dynamic test](https://www.vogella.com/tutorials/JUnit/article.html#write-a-dynamic-test)

Create the following test class.

**package** **com.vogella.unittest.converter;**

**import** **java.util.Arrays;**

**import** **java.util.stream.Stream;**

**import** **org.junit.jupiter.api.DynamicTest;**

**import** **org.junit.jupiter.api.TestFactory;**

**class** **ConverterUtilTest** **{**

**int[][]** celsiusFahrenheitMapping **=** **new** **int[][]** **{** **{** **10,** **50** **},** **{** **40,** **104** **},** **{** **0,** **32** **}** **};**

**@TestFactory**

**Stream<DynamicTest>** ensureThatCelsiumConvertsToFahrenheit**()** **{**

**return** **Arrays.**stream**(**celsiusFahrenheitMapping**).**map**(**entry **->** **{**

*// access celcius and fahrenheit from entry*

**int** celsius **=** entry**[0];**

**int** fahrenheit **=** entry**[1];**

**return** **null;**

*// return a dynamicTest which checks that that the convertion from celcius to*

*// fahrenheit is correct*

**});**

**}**

**Stream<DynamicTest>** ensureThatFahrenheitToCelsiumConverts**()** **{**

**return** **null;**

*// TODO Write a similar test fahrenheit to celsius*

**}**

**}**

Fix all the failing test, unfortunately the test specification is not very good. Try to write reasonable tests which fit the method name.

Show Solution

### [11.3. Verify](https://www.vogella.com/tutorials/JUnit/article.html#verify-3)

Run your new test via the IDE and ensure that you have 6 tests running succesfull.y

Verify that your code compiles and your test are running via the command line either with ./gradlew test`or with the `mvn clean verify depending on your build system.

## [12. Exercise: Testing with multiple input parameter](https://www.vogella.com/tutorials/JUnit/article.html#exercise-testing-with-multiple-input-parameter)

Dynamic tests are included in the regular JUnit 5 library, which you already included. To use parameters in your tests, you have to add the junit-jupiter-params library.

### [12.1. Add dependency](https://www.vogella.com/tutorials/JUnit/article.html#add-dependency)

If you are using Maven add the following dependency to junit-jupiter-params to your Maven pom file.

org.junit.jupiter

junit-jupiter-params

5.7.2

test

If you are using Gradle add the following to your build.gradle file

implementation 'org.junit.jupiter:junit-jupiter-params:5.7.2'

### [12.2. Write a parameterized test](https://www.vogella.com/tutorials/JUnit/article.html#write-a-parameterized-test)

Review the following code:

**package** **com.vogella.unittest.converter;**

**import** **static** org**.**junit**.**jupiter**.**api**.**Assertions**.**assertEquals**;**

**import** **org.junit.jupiter.params.ParameterizedTest;**

**import** **org.junit.jupiter.params.provider.MethodSource;**

**class** **ParameterizedExampleTest** **{**

**static** **int[][]** data**()** **{**

**return** **new** **int[][]** **{** **{** **1** **,** **2,** **2** **},** **{** **5,** **3,** **15** **},** **{** **121,** **4,** **484** **}** **};**

**}**

**@ParameterizedTest(**name **=** **"{index} called with: {0}")**

**@MethodSource(**value **=** **"data")**

**void** testWithStringParameter**(int[]** data**)** **{**

**MyClass** tester **=** **new** **MyClass();**

**int** m1 **=** data**[0];**

**int** m2 **=** data**[1];**

**int** expected **=** data**[2];**

assertEquals**(**expected**,** tester**.**multiply**(**m1**,** m2**));**

**}**

*// class to be tested*

**class** **MyClass** **{**

**public** **int** multiply**(int** i**,** **int** j**)** **{**

**return** i **\*** j**;**

**}**

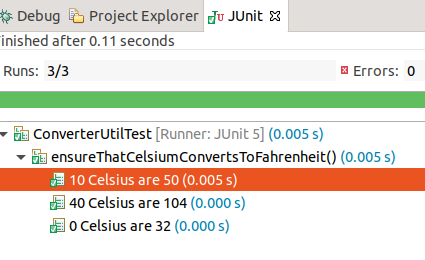
**}**

**}**

Create a new test method in ConverterUtilTest which also uses a parameterized test.

### [12.3. Verify](https://www.vogella.com/tutorials/JUnit/article.html#verify-4)

Run your new test via the IDE.



Verify that your code compiles and your test are running via the command line with the ./gradlew test or mvn clean verify command based on your build system.

### [12.4. Add more options](https://www.vogella.com/tutorials/JUnit/article.html#add-more-options)

ParameterizedTest are very flexible in sense of their input. The following lists a few more. Add these to your test and run the tests again.

**@ParameterizedTest**

**@ValueSource(**strings **=** **{** **"WINDOW",** **"Microsoft Windows [Version 10.?]"** **})**

**void** ensureWindowsStringContainWindow**(String** name**)** **{**

assertTrue**(**name**.**toLowerCase**().**contains**("window"));**

**}**

**@DisplayName("A negative value for year is not supported by the leap year computation.")**

**@ParameterizedTest(**name **=** **"For example, year {0} is not supported.")**

**@ValueSource(**ints **=** **{** **-1,** **-4** **})**

**void** ensureYear**(int** year**)** **{**

assertTrue**(**year **<** **0);**

**}**

**@ParameterizedTest(**name **=** **"{0} \* {1} = {2}")**

**@CsvSource({** **"0, 1, 0",** **"1, 2, 2",** **"49, 50, 2450",** **"1, 100, 100"** **})**

**void** add**(int** first**,** **int** second**,** **int** expectedResult**)** **{**

**MyClass** calculator **=** **new** **MyClass();**

assertEquals**(**expectedResult**,** calculator**.**multiply**(**first**,** second**),**

**()** **->** first **+** **" \* "** **+** second **+** **" should equal "** **+** expectedResult**);**

**}**

## [13. Exercise: Using the @TempDir annotation to create temporary files and paths](https://www.vogella.com/tutorials/JUnit/article.html#exercise-using-the-tempdir-annotation-to-create-temporary-files-and-paths)

In this exercise you learn how to use the @TempDir annotation to let JUnit 5 create files and paths on request in your test and to automatically remove them after the test.

|  |  |
| --- | --- |
|  | Java 11 API for creating files: |

### [13.1. Create class under test](https://www.vogella.com/tutorials/JUnit/article.html#create-class-under-test)

Create the following class

**package** **com.vogella.unittest.file;**

**import** **java.io.IOException;**

**import** **java.nio.charset.StandardCharsets;**

**import** **java.nio.file.Files;**

**import** **java.nio.file.Path;**

**import** **java.nio.file.StandardOpenOption;**

**public** **class** **FileWriter** **{**

**private** FileWriter**()** **{**

**}**

**public** **static** **void** createFile**(Path** path**)** **{**

**try** **{**

**Files.**write**(**path**,** **"".**getBytes**(StandardCharsets.**UTF\_8**),** **StandardOpenOption.**CREATE**,**

**StandardOpenOption.**TRUNCATE\_EXISTING**);**

**}** **catch** **(IOException** e**)** **{**

e**.**printStackTrace**();**

**}**

**}**

**public** **static** **void** appendFile**(Path** path**,** **String** content**)** **throws** **IOException** **{**

*// image more logic here...*

**Files.**writeString**(**path**,** content**,** **StandardOpenOption.**APPEND**);**

**}**

**}**

### [13.2. Write tests](https://www.vogella.com/tutorials/JUnit/article.html#write-tests-2)

Using the @TempDir annotation, create unit which test named FileWriterTest for the following:

* Ensure that the Path given to you by the @TempDir annotation if writable
* Ensure that a appending to a file with FileWriter.appendFile which has not yet been created with FileWriter.createFile throws an exception
* Ensure that you can write to the file once you created it

HINT:

**@Test**

**void** ensureThatPathFromTempDirISWritable**(@TempDir** **Path** path**)** **{**

*// Check if the path created by the TempDir extension is writable*

*// Check `Files` API for this*

**}**

#### [**13.2.1. Solution**](https://www.vogella.com/tutorials/JUnit/article.html#solution-4)

Solution

## [14. Exercise: Testing for annotations](https://www.vogella.com/tutorials/JUnit/article.html#exercise-testing-for-annotations)

In this exercise you write test to check class under test for certain annotations.

### [14.1. Add dependency to @Inject](https://www.vogella.com/tutorials/JUnit/article.html#add-dependency-to-inject)

If you have not yet done this, add a dependency to javax.inject.

Maven:

**<dependency>**

**<groupId>**javax.inject**</groupId>**

**<artifactId>**javax.inject**</artifactId>**

**<version>**1**</version>**

**</dependency>**

Gradle:

implementation **'javax.inject:javax.inject:1'**

### [14.2. Create class under test](https://www.vogella.com/tutorials/JUnit/article.html#create-class-under-test-2)

Create the following class

**package** **com.vogella.unittest.di;**

**import** **javax.inject.Inject;**

**public** **class** **Service** **{**

**@Inject**

**String** s**;**

**@Inject**

**public** Service**()** **{**

**}**

**@Inject**

**public** Service**(String** s**)** **{**

**this.**s **=** s**;**

**}**

**}**

t validates that the Servic === Write tests

Write a test that validates that the Service class only has one constructor annotated with @Inject.

HINT:

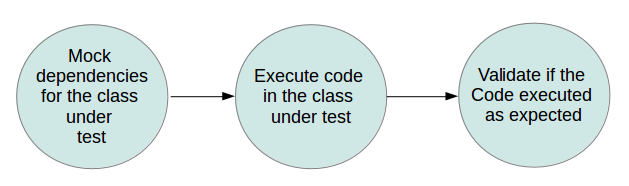
* The class has a `getConstructors method.
* The Constructor has a method getAnnotation

Mockito:

Mockito is a popular open source framework for mocking objects in software test. Using Mockito greatly simplifies the development of tests for classes with external dependencies.

A mock object is a dummy implementation for an interface or a class. It allows to define the output of certain method calls. They typically record the interaction with the system and tests can validate that.

This allows you to simplify the test setup.



Mockito records the interaction with mock and allows you to check if the mock object was used correct, e.g. if a certain method has been called on the mock. This allows you to implement behavior testing instead of only testing the result of method calls.

*// state testing*

testSort**()** **{**

testList **=** **[1,** **7,** **3,** **8,** **2]**

**MySorter.**sort**(**testList**)**

**assert** testList equals **[1,** **2,** **3,** **7,** **8]**

**}**

*// incorrect would be behavior testing*

*// the following tests internal of the implementation*

testSort**()** **{**

testList **=** **[1,** **7,** **3,** **8,** **2]**

**MySorter.**sort**(**testList**)**

**assert** that compare**(1,** **2)** was called once

**assert** that compare**(1,** **3)** was not called

**assert** that compare**(2,** **3)** was called once

**....**

**}**

## [3. Adding Mockito to a project](https://www.vogella.com/tutorials/Mockito/article.html#mockito_installation)

Using the Mockito libraries should be done with a modern dependency system like Maven or Gradle. All modern IDEs (Eclipse, Visual Studio Code, IntelliJ) support both Maven and Gradle.

The following contains detailed descriptions for your environment, pick the one which is relevant for you. The latest version of Mockito can be found via <https://search.maven.org/artifact/org.mockito/mockito-core>.

## [4. Creating mock objects with the Mockito API](https://www.vogella.com/tutorials/Mockito/article.html#mockitousage)

Mockito provides several methods to create mock objects:

* Using the @ExtendWith(MockitoExtension.class) extension for JUnit 5 in combination with the @Mock annotation on fields
* Using the static mock() method.
* Using the @Mock annotation.

If you use the @Mock annotation, you must trigger the initialization of the annotated fields. The MockitoExtension does this by calling the static method MockitoAnnotations.initMocks(this).

For example, consider the following data model.

**package** **com.vogella.junit5;**

**public** **class** **Database** **{**

**public** **boolean** isAvailable**()** **{**

*// currently not implemented, as this is just demo used in a software test*

**return** **false;**

**}**

**public** **int** getUniqueId**()** **{**

**return** **42;**

**}**

**}**

**package** **com.vogella.junit5;**

**public** **class** **Service** **{**

**private** **Database** database**;**

**public** Service**(Database** database**)** **{**

**this.**database **=** database**;**

**}**

**public** **boolean** query**(String** query**)** **{**

**return** database**.**isAvailable**();**

**}**

**@Override**

**public** **String** toString**()** **{**

**return** **"Using database with id: "** **+** **String.**valueOf**(**database**.**getUniqueId**());**

**}**

**}**

A unit test using Mockito which mocks the Database object could look like the following.

**package** **com.vogella.junit5;**

**import** **static** org**.**junit**.**jupiter**.**api**.**Assertions**.**assertNotNull**;**

**import** **static** org**.**junit**.**jupiter**.**api**.**Assertions**.**assertTrue**;**

**import** **static** org**.**mockito**.**Mockito**.**when**;**

**import** **org.junit.jupiter.api.Test;**

**import** **org.junit.jupiter.api.extension.ExtendWith;**

**import** **org.mockito.Mock;**

**import** **org.mockito.junit.jupiter.MockitoExtension;**

**@ExtendWith(MockitoExtension.**class**)**

**class** **ServiceTest** **{**

**@Mock**

**Database** databaseMock**;**

**@Test**

**public** **void** testQuery**()** **{**

assertNotNull**(**databaseMock**);**

when**(**databaseMock**.**isAvailable**()).**thenReturn**(true);**

**Service** t **=** **new** **Service(**databaseMock**);**

**boolean** check **=** t**.**query**("\* from t");**

assertTrue**(**check**);**

**}**

**}**

|  |  |
| --- | --- |
|  | Tells Mockito to create the mocks based on the @Mock annotation, this requires JUnit 5, if you an oder version of JUnit, call Mock.init() in your setup method |
|  | Tells Mockito to mock the databaseMock instance |
|  | Configure the Mock to return true when its isAvailable method is called, see later for more options |
|  | Executes some code of the class under test |
|  | Asserts that the method call returned true |
|  | *Static imports*  Mockito provides a lot of static methods for mock and asserts. By adding the org.mockito.Mockito.\*; static import, you can use these methods directly in your tests. Static imports allow you to call static members, i.e., methods and fields of a class directly without specifying the class.  Using static imports also greatly improves the readability of your test code. |

## [5. Configuring the return values of methods calls on the mock objects](https://www.vogella.com/tutorials/Mockito/article.html#configuring-the-return-values-of-methods-calls-on-the-mock-objects)

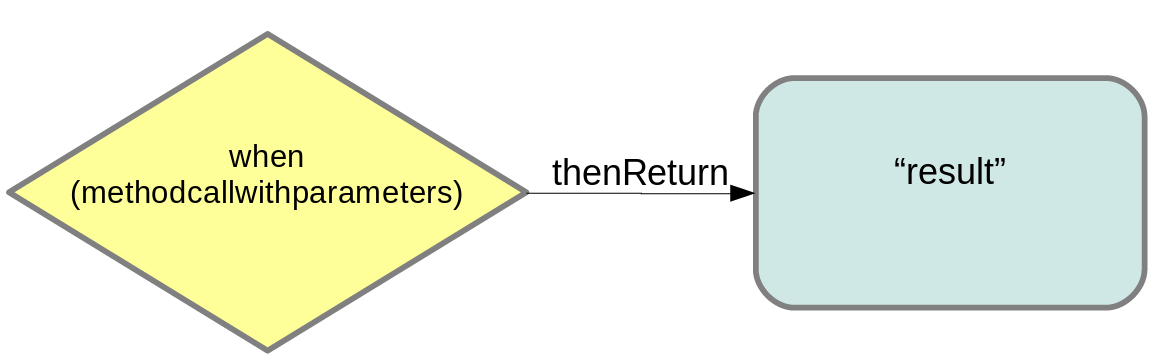
Mockito allows to configure the return values of methods which are called on the mock via a fluent API. Unspecified method calls return "empty" values:

* null for objects
* 0 for numbers
* false for boolean
* empty collections for collections
* …​

|  |  |
| --- | --- |
|  | The following assert statements are only for demonstration purposes, a real test would use the mocks to unit test another functionality. |

### [5.1. Using **when().thenReturn()** and **when().thenThrow()**](https://www.vogella.com/tutorials/Mockito/article.html#using-when-thenreturn-and-when-thenthrow)

Mocks can return different values depending on arguments passed into a method. The when(…​.).thenReturn(…​.) method chain is used to specify a return value for a method call with pre-defined parameters.



You also can use methods like anyString or anyInt to define that dependent on the input type a certain value should be returned.

If you specify more than one value, they are returned in the order of specification, until the last one is used. Afterwards the last specified value is returned.

The following demonstrates the usage of when(…​.).thenReturn(…​.).

**package** **com.vogella.junit5;**

**import** **static** org**.**junit**.**jupiter**.**api**.**Assertions**.**assertEquals**;**

**import** **static** org**.**mockito**.**Mockito**.**when**;**

**import** **org.junit.jupiter.api.Test;**

**import** **org.junit.jupiter.api.extension.ExtendWith;**

**import** **org.mockito.Mock;**

**import** **org.mockito.junit.jupiter.MockitoExtension;**

**@ExtendWith(MockitoExtension.**class**)**

**class** **ServiceDatabaseIdTest** **{**

**@Mock**

**Database** databaseMock**;**

**@Test**

**void** ensureMockitoReturnsTheConfiguredValue**()** **{**

*// define return value for method getUniqueId()*

when**(**databaseMock**.**getUniqueId**()).**thenReturn**(42);**

**Service** service **=** **new** **Service(**databaseMock**);**

*// use mock in test....*

assertEquals**(**service**.**toString**(),** **"Using database with id: 42");**

**}**

**}**

Other examples which demonstrates the configuration of Mockito are in the following listing. These are not real test, the test only validating the Mockito configuration.

**package** **com.vogella.junit5;**

**import** **static** org**.**junit**.**jupiter**.**api**.**Assertions**.**assertEquals**;**

**import** **static** org**.**mockito**.**ArgumentMatchers**.**isA**;**

**import** **static** org**.**mockito**.**Mockito**.**when**;**

**import** **java.util.Iterator;**

**import** **org.junit.jupiter.api.Test;**

**import** **org.junit.jupiter.api.extension.ExtendWith;**

**import** **org.mockito.Mock;**

**import** **org.mockito.junit.jupiter.MockitoExtension;**

**@ExtendWith(MockitoExtension.**class**)**

**class** **MockitoWhenExampleTest** **{**

**@Mock**

**Iterator<String>** i**;**

**Comparable<String>** c**;**

*// demonstrates the return of multiple values*

**@Test**

**void** testMoreThanOneReturnValue**()** **{**

when**(**i**.**next**()).**thenReturn**("Mockito").**thenReturn**("rocks");**

**String** result **=** i**.**next**()** **+** **" "** **+** i**.**next**();**

*// assert*

assertEquals**("Mockito rocks",** result**);**

**}**

*// this test demonstrates how to return values based on the input*

*// and that @Mock can also be used for a method parameter*

**@Test**

**void** testReturnValueDependentOnMethodParameter**(@Mock** **Comparable<String>** c**)** **{**

when**(**c**.**compareTo**("Mockito")).**thenReturn**(1);**

when**(**c**.**compareTo**("Eclipse")).**thenReturn**(2);**

*//assert*

assertEquals**(1,** c**.**compareTo**("Mockito"));**

assertEquals**(2,** c**.**compareTo**("Eclipse"));**

**}**

*// return a value based on the type of the provide parameter*

**@Test**

**void** testReturnValueInDependentOnMethodParameter2**(@Mock** **Comparable<Integer>** c **)** **{**

when**(**c**.**compareTo**(**isA**(Integer.**class**))).**thenReturn**(0);**

*//assert*

assertEquals**(0,** c**.**compareTo**(Integer.**valueOf**(4)));**

**}**

**}**

The when(…​.).thenReturn(…​.) method chain can be used to throw an exception.

**package** **com.vogella.junit5;**

**import** **static** org**.**junit**.**jupiter**.**api**.**Assertions**.**assertEquals**;**

**import** **static** org**.**junit**.**jupiter**.**api**.**Assertions**.**assertThrows**;**

**import** **static** org**.**mockito**.**Mockito**.**when**;**

**import** **java.util.Properties;**

**import** **org.junit.jupiter.api.Test;**

**import** **org.mockito.Mockito;**

**class** **MockitoThrowsTest** **{**

*// demonstrates the configuration of a throws with Mockito*

*// not a read test, just "testing" Mockito behavior*

**@Test**

**void** testMockitoThrows**()** **{**

**Properties** properties **=** **Mockito.**mock**(Properties.**class**);**

when**(**properties**.**get**(Mockito.**anyString**())).**thenThrow**(new** **IllegalArgumentException("Stuff"));**

**Throwable** exception **=** assertThrows**(IllegalArgumentException.**class**,** **()** **->** properties**.**get**("A"));**

assertEquals**("Stuff",** exception**.**getMessage**());**

**}**

**}**

### [5.2. **doReturn when** and **doThrow when**](https://www.vogella.com/tutorials/Mockito/article.html#doreturn-when-and-dothrow-when)

The doReturn(…​).when(…​) method configuration can be used to configure the reply of a mocked method call. This is similar to when(…​.).thenReturn(…​.).

As when(…​.).thenReturn(…​.) is easier to read, you should prefer using it. But doReturn can be useful if you are using spies, e.g. with @Spy, in Mockito. In case of a spy Mockito uses the underlying object and when(…​.).thenReturn(…​.) will call the underlying method which may create side effects. If these side effects are not desired, use doReturn, e.g. if you spy on a list and a certain operation would result in an undesired exception from the original object, like requesting the element in position 2 of an empty list.

**package** **com.vogella.junit5;**

**import** **static** org**.**junit**.**jupiter**.**api**.**Assertions**.**assertEquals**;**

**import** **static** org**.**mockito**.**Mockito**.**doReturn**;**

**import** **static** org**.**mockito**.**Mockito**.**spy**;**

**import** **java.util.Properties;**

**import** **org.junit.jupiter.api.Test;**

**class** **MockitoSpyWithListTest** **{**

*// demonstrates of the spy function*

**@Test**

**void** ensureSpyForListWorks**()** **{**

**var** list **=** **new** **ArrayList<String>();**

**var** spiedList **=** spy**(**list**);**

doReturn**("42").**when**(**spiedList**).**get**(99);**

**String** value **=** **(String)** spiedList**.**get**(99);**

assertEquals**("42",** value**);**

**}**

**}**

The doThrow variant can be used for methods which return void to throw an exception. This usage is demonstrated by the following code snippet.

**package** **com.vogella.junit5;**

**import** **static** org**.**junit**.**jupiter**.**api**.**Assertions**.**assertThrows**;**

**import** **static** org**.**mockito**.**Mockito**.**doThrow**;**

**import** **static** org**.**mockito**.**Mockito**.**mock**;**

**import** **java.io.IOException;**

**import** **java.io.OutputStream;**

**import** **java.io.OutputStreamWriter;**

**import** **org.junit.jupiter.api.Test;**

**class** **MockitoThrowsTest** **{**

**@Test**

**public** **void** testForIOException**()** **throws** **IOException** **{**

*// create an configure mock*

**OutputStream** mockStream **=** mock**(OutputStream.**class**);**

doThrow**(new** **IOException()).**when**(**mockStream**).**close**();**

*// use mock*

**OutputStreamWriter** streamWriter **=** **new** **OutputStreamWriter(**mockStream**);**

assertThrows**(IOException.**class**,** **()** **->** streamWriter**.**close**());**

**}**

**}**

## [6. Wrapping Java objects with Spy](https://www.vogella.com/tutorials/Mockito/article.html#mockito_spy)

@Spy or the spy() method can be used to wrap a real object. Every call, unless specified otherwise, is delegated to the object.

**package** **com.vogella.mockito.spy;**

**import** **static** org**.**junit**.**jupiter**.**api**.**Assertions**.**assertEquals**;**

**import** **static** org**.**mockito**.**Mockito**.**doReturn**;**

**import** **java.util.LinkedList;**

**import** **java.util.List;**

**import** **org.junit.jupiter.api.BeforeEach;**

**import** **org.junit.jupiter.api.Test;**

**import** **org.junit.jupiter.api.extension.ExtendWith;**

**import** **org.mockito.Spy;**

**import** **org.mockito.junit.jupiter.MockitoExtension;**

**@ExtendWith(MockitoExtension.**class**)**

**class** **MockitoSpyTest** **{**

**@Spy**

**List<String>** spy **=** **new** **LinkedList<>();**

**@BeforeEach**

**void** setup**()** **{**

*// Alternative way of creating a spy*

*// List<String> list = new LinkedList<>();*

*// List<String> spy = spy(list);*

**}**

**@Test**

**void** testLinkedListSpyCorrect**()** **{**

*// when(spy.get(0)).thenReturn("foo");*

*// would not work as the delegate it called so spy.get(0)*

*// throws IndexOutOfBoundsException (list is still empty)*

*// you have to use doReturn() for stubbing*

doReturn**("foo").**when**(**spy**).**get**(0);**

assertEquals**("foo",** spy**.**get**(0));**

**}**

**}**

### [6.1. Verify the calls on the mock objects](https://www.vogella.com/tutorials/Mockito/article.html#mockito_verify)

Mockito keeps track of all the method calls and their parameters to the mock object. You can use the verify() method on the mock object to verify that the specified conditions are met. For example, you can verify that a method has been called with certain parameters. This kind of testing is sometimes called behavior testing. Behavior testing does not check the result of a method call, but it checks that a method is called with the right parameters.

**package** **com.vogella.junit5;**

**import** **static** org**.**mockito**.**Mockito**.**atLeast**;**

**import** **static** org**.**mockito**.**Mockito**.**atLeastOnce**;**

**import** **static** org**.**mockito**.**Mockito**.**never**;**

**import** **static** org**.**mockito**.**Mockito**.**times**;**

**import** **static** org**.**mockito**.**Mockito**.**verify**;**

**import** **static** org**.**mockito**.**Mockito**.**verifyNoMoreInteractions**;**

**import** **static** org**.**mockito**.**Mockito**.**when**;**

**import** **org.junit.jupiter.api.Test;**

**import** **org.junit.jupiter.api.extension.ExtendWith;**

**import** **org.mockito.ArgumentMatchers;**

**import** **org.mockito.Mock;**

**import** **org.mockito.junit.jupiter.MockitoExtension;**

**@ExtendWith(MockitoExtension.**class**)**

**public** **class** **MockitoVerifyTest** **{**

**@Test**

**public** **void** testVerify**(@Mock** **Database** database**)** **{**

*// create and configure mock*

when**(**database**.**getUniqueId**()).**thenReturn**(43);**

*// call method testing on the mock with parameter 12*

database**.**setUniqueId**(12);**

database**.**getUniqueId**();**

database**.**getUniqueId**();**

*// now check if method testing was called with the parameter 12*

verify**(**database**).**setUniqueId**(ArgumentMatchers.**eq**(12));**

*// was the method called twice?*

verify**(**database**,** times**(2)).**getUniqueId**();**

*// other alternatives for verifiying the number of method calls for a method*

verify**(**database**,** never**()).**isAvailable**();**

verify**(**database**,** never**()).**setUniqueId**(13);**

verify**(**database**,** atLeastOnce**()).**setUniqueId**(12);**

verify**(**database**,** atLeast**(2)).**getUniqueId**();**

*// more options are*

*// times(numberOfTimes)*

*// atMost(numberOfTimes)*

*// This let's you check that no other methods where called on this object.*

*// You call it after you have verified the expected method calls.*

verifyNoMoreInteractions**(**database**);**

**}**

**}**

In case you do not care about the value, use the anyX, e.g., anyInt, anyString(), or any(YourClass.class) methods.

## [7. Using @InjectMocks for dependency injection via Mockito](https://www.vogella.com/tutorials/Mockito/article.html#mockito_dependencyinjection)

You also have the @InjectMocks annotation which tries to do constructor, method or field dependency injection of mock objects in to other type. It does not require @Inject to be present and also does not try to be a full features dependency injection framework but is helpful to wire up objects to be tested.

For example, assume you have a class named ArticleManager withe the following constructor:

**public** ArticleManager**(User** user**,** **ArticleDatabase** database**)** **{**

*// more code...*

**}**

This class can be constructed via Mockito. The parameters to the constructor could be supplied via mock objects as demonstrated by the following code snippet.

**package** **com.vogella.junit5;**

**import** **org.junit.jupiter.api.Test;**

**import** **org.junit.jupiter.api.extension.ExtendWith;**

**import** **org.mockito.InjectMocks;**

**import** **org.mockito.Mock;**

**import** **org.mockito.Mockito;**

**import** **org.mockito.junit.jupiter.MockitoExtension;**

**@ExtendWith(MockitoExtension.**class**)**

**class** **ArticleManagerTest** **{**

**@Mock**

**ArticleDatabase** database**;**

**@Mock**

**User** user**;**

**@InjectMocks**

**private** **ArticleManager** manager**;**

**@Test**

**void** shouldDoSomething**()** **{**

*// TODO perform some tests with this managers*

**}**

**}**

|  |  |
| --- | --- |
|  | creates an instance of ArticleManager and injects the mocks into it |

You find a full example of the usage of @InjectMocks in the exercises. Mockito can inject mocks either via constructor injection, setter injection, or property injection and in this order. So if ArticleManager would have a constructor that would only take User and setters for both fields, only the mock for User would be injected.

## [8. Capturing the arguments](https://www.vogella.com/tutorials/Mockito/article.html#capturing-the-arguments)

The ArgumentCaptor class allows to access the arguments of method calls during the verification. This allows to capture these arguments of method calls and to use them for tests.

To run this example you need to add [hamcrest-library](https://mvnrepository.com/artifact/org.hamcrest/hamcrest-library) to your project.

**package** **com.vogella.junit5;**

**import** **static** org**.**hamcrest**.**MatcherAssert**.**assertThat**;**

**import** **static** org**.**hamcrest**.**Matchers**.**hasItem**;**

**import** **static** org**.**mockito**.**Mockito**.**verify**;**

**import** **java.util.Arrays;**

**import** **java.util.List;**

**import** **org.junit.jupiter.api.Test;**

**import** **org.junit.jupiter.api.extension.ExtendWith;**

**import** **org.mockito.ArgumentCaptor;**

**import** **org.mockito.Captor;**

**import** **org.mockito.Mock;**

**import** **org.mockito.junit.jupiter.MockitoExtension;**

**@ExtendWith(MockitoExtension.**class**)**

**public** **class** **MockitoArgumentCaptureTest** **{**

**@Captor**

**private** **ArgumentCaptor<List<String>>** captor**;**

**@Test**

**public** **final** **void** shouldContainCertainListItem**(@Mock** **List<String>** mockedList**)** **{**

**var** asList **=** **Arrays.**asList**("someElement\_test",** **"someElement");**

mockedList**.**addAll**(**asList**);**

verify**(**mockedList**).**addAll**(**captor**.**capture**());**

**List<String>** capturedArgument **=** captor**.**getValue**();**

assertThat**(**capturedArgument**,** hasItem**("someElement"));**

**}**

**}**

## [9. Using Answers for complex mocks](https://www.vogella.com/tutorials/Mockito/article.html#mockito_answers)

thenReturn returns a predefined value every time. With an Answer object you can calculate a response based on the arguments given to your stubbed method.

This can be useful if your stubbed method is supposed to call a function on one of the arguments or if your method is supposed to return the first argument to allow method chaining. There exists a static method for the latter. Also note that there a different ways to configure an answer:

**import** **static** org**.**mockito**.**AdditionalAnswers**.**returnsFirstArg**;**

**@Test**

**public** **final** **void** answerTest**()** **{**

*// with doAnswer():*

doAnswer**(**returnsFirstArg**()).**when**(**list**).**add**(**anyString**());**

*// with thenAnswer():*

when**(**list**.**add**(**anyString**())).**thenAnswer**(**returnsFirstArg**());**

*// with then() alias:*

when**(**list**.**add**(**anyString**())).**then**(**returnsFirstArg**());**

**}**

Or if you need to do a callback on your argument:

**@Test**

**public** **final** **void** callbackTest**()** **{**

**ApiService** service **=** mock**(ApiService.**class**);**

when**(**service**.**login**(**any**(Callback.**class**))).**thenAnswer**(**i **->** **{**

**Callback** callback **=** i**.**getArgument**(0);**

callback**.**notify**("Success");**

**return** **null;**

**});**

**}**

It is even possible to mock a persistence service like an DAO, but you should consider creating a fake class instead of a mock if your Answers become too complex.

**List<User>** userMap **=** **new** **ArrayList<>();**

**UserDao** dao **=** mock**(UserDao.**class**);**

when**(**dao**.**save**(**any**(User.**class**))).**thenAnswer**(**i **->** **{**

**User** user **=** i**.**getArgument**(0);**

userMap**.**add**(**user**.**getId**(),** user**);**

**return** **null;**

**});**

when**(**dao**.**find**(**any**(Integer.**class**))).**thenAnswer**(**i **->** **{**

**int** id **=** i**.**getArgument**(0);**

**return** userMap**.**get**(**id**);**

**});**

## [10. More on Mockito](https://www.vogella.com/tutorials/Mockito/article.html#more-on-mockito)

### [10.1. Mocking final classes and static methods](https://www.vogella.com/tutorials/Mockito/article.html#mockstaticandfinal)

Since the mockito-inline library replaced the mockito-core library it is possible to mock final classes and static methods.

For example, if you have the following final class:

**final** **class** **FinalClass** **{**

**public** **final** **String** finalMethod**()** **{** **return** **"something";** **}**

**}**

You can mock it via the following code:

**package** **com.vogella.junit5;**

**import** **static** org**.**junit**.**jupiter**.**api**.**Assertions**.**assertNotNull**;**

**import** **org.junit.jupiter.api.Test;**

**import** **org.junit.jupiter.api.extension.ExtendWith;**

**import** **org.mockito.Mock;**

**import** **org.mockito.MockedStatic;**

**import** **org.mockito.Mockito;**

**import** **org.mockito.junit.jupiter.MockitoExtension;**

**@ExtendWith(MockitoExtension.**class**)**

**public** **class** **MockitoMockFinal** **{**

**@Test**

**public** **void** testMockFinal**(@Mock** **FinalClass** finalMocked**)** **{**

assertNotNull**(**finalMocked**);**

**}**

**@Test**

**public** **void** testMockFinalViaMockStatic**()** **{**

**MockedStatic<FinalClass>** mockStatic **=** **Mockito.**mockStatic**(FinalClass.**class**);**

assertNotNull**(**mockStatic**);**

**}**

**}**

Mockito also allows to mock static methods.

**package** **com.vogella.junit5;**

**public** **class** **Utility** **{**

**public** **static** **String** getDatabaseConnection**(String** url**)** **{**

**return** **"http:///production/"** **+** url**;**

**}**

**}**

**package** **com.vogella.junit5;**

**import** **static** org**.**junit**.**jupiter**.**api**.**Assertions**.**assertEquals**;**

**import** **org.junit.jupiter.api.Test;**

**import** **org.mockito.MockedStatic;**

**import** **org.mockito.Mockito;**

**class** **MyUtilsTest** **{**

**@Test**

**void** shouldMockStaticMethod**()** **{**

**MockedStatic<FinalClass>** mockStatic **=** **Mockito.**mockStatic**(FinalClass.**class**);**

**try** **(MockedStatic<Utility>** mockedStatic **=** **Mockito.**mockStatic**(Utility.**class**))** **{**

mockedStatic**.**when**(()** **->** **Utility.**getDatabaseConnection**(Mockito.**eq**("test"))).**thenReturn**("testing");**

mockedStatic**.**when**(()** **->** **Utility.**getDatabaseConnection**(Mockito.**eq**("prod"))).**thenReturn**("production");**

**String** result1 **=** **Utility.**getDatabaseConnection**("test");**

assertEquals**("testing",** result1**);**

**String** result2 **=** **Utility.**getDatabaseConnection**("prod");**

assertEquals**("production",** result2**);**

**}**

**}**

**}**

### [10.2. Clean test code with the help of the strict stubs rule](https://www.vogella.com/tutorials/Mockito/article.html#clean-test-code-with-the-help-of-the-strict-stubs-rule)

The strict stubs rule helps you to keep your test code clean and checks for common oversights. It adds the following:

* test fails early when a stubbed method gets called with different arguments than what it was configured for (with PotentialStubbingProblem exception).
* test fails when a stubbed method isn’t called (with UnnecessaryStubbingException exception).
* org.mockito.Mockito.verifyNoMoreInteractions(Object) also verifies that all stubbed methods have been called during the test

**@Test**

**public** **void** withoutStrictStubsTest**()** **throws** **Exception** **{**

**DeepThought** deepThought **=** mock**(DeepThought.**class**);**

when**(**deepThought**.**getAnswerFor**("Ultimate Question of Life, The Universe, and Everything")).**thenReturn**(42);**

when**(**deepThought**.**otherMethod**("some mundane thing")).**thenReturn**(null);**

**System.**out**.**println**(**deepThought**.**getAnswerFor**("Six by nine"));**

assertEquals**(42,** deepThought**.**getAnswerFor**("Ultimate Question of Life, The Universe, and Everything"));**

verify**(**deepThought**,** times**(1)).**getAnswerFor**("Ultimate Question of Life, The Universe, and Everything");**

**}**

*// activate the strict subs rule*

**@Rule** **public** **MockitoRule** rule **=** **MockitoJUnit.**rule**().**strictness**(Strictness.**STRICT\_STUBS**);**

**@Test**

**public** **void** withStrictStubsTest**()** **throws** **Exception** **{**

**DeepThought** deepThought **=** mock**(DeepThought.**class**);**

when**(**deepThought**.**getAnswerFor**("Ultimate Question of Life, The Universe, and Everything")).**thenReturn**(42);**

*// this fails now with an UnnecessaryStubbingException since it is never called in the test*

when**(**deepThought**.**otherMethod**("some mundane thing")).**thenReturn**(null);**

*// this will now throw a PotentialStubbingProblem Exception since we usually don't want to call methods on mocks without configured behavior*

deepThought**.**someMethod**();**

assertEquals**(42,** deepThought**.**getAnswerFor**("Ultimate Question of Life, The Universe, and Everything"));**

*// verifyNoMoreInteractions now automatically verifies that all stubbed methods have been called as well*

verifyNoMoreInteractions**(**deepThought**);**

**}**

## [11. Exercise: Create a new Maven or Gradle project with Mockito support](https://www.vogella.com/tutorials/Mockito/article.html#exercise-create-a-new-maven-or-gradle-project-with-mockito-support)

Create a new Maven or Gradle project named com.vogella.mockito.

Add the required libraries to your project depending on your build system.

Using Maven to add Mockito to your project

|  |  |
| --- | --- |
|  |  |

Using Gradle to add Mockito to your project

You can now use Mockito in your project.

## [12. Exercise: Testing an API with Mockito and JUnit 5](https://www.vogella.com/tutorials/Mockito/article.html#exercise-testing-an-api-with-mockito-and-junit-5)

### [12.1. Create an AudioManager API](https://www.vogella.com/tutorials/Mockito/article.html#create-an-audiomanager-api)

Create the following classes which will be used in the following tests.

**package** **com.vogella.mockito.audio;**

**public** **enum** **RINGER\_MODE** **{**

**RINGER\_MODE\_NORMAL,** **RINGER\_MODE\_SILENT**

**}**

**package** **com.vogella.mockito.audio;**

**public** **class** **AudioManager** **{**

**private** **int** volume **=** **50;**

**private** **RINGER\_MODE** mode **=** **RINGER\_MODE.**RINGER\_MODE\_SILENT**;**

**public** **RINGER\_MODE** getRingerMode**()** **{**

**return** mode**;**

**}**

**public** **int** getStreamMaxVolume**()** **{**

**return** volume**;**

**}**

**public** **void** setStreamVolume**(int** max**)** **{**

volume **=** max**;**

**}**

**public** **void** makeReallyLoad**()** **{**

**if** **(**mode**.**equals**(RINGER\_MODE.**RINGER\_MODE\_NORMAL**))** **{**

setStreamVolume**(100);**

**}**

**}**

**}**

**package** **com.vogella.mockito.audio;**

**public** **class** **MyApplication** **{**

**public** **int** getNumberOfThreads**()** **{**

**return** **5;**

**}**

**}**

**package** **com.vogella.mockito.audio;**

**public** **class** **ConfigureThreadingUtil** **{**

**public** **static** **void** configureThreadPool**(MyApplication** app**){**

**int** numberOfThreads **=** app**.**getNumberOfThreads**();**

*// TODO use information to configure the thread pool*

**}**

**}**

**package** **com.vogella.mockito.audio;**

**public** **class** **VolumeUtil** **{**

**public** **static** **void** maximizeVolume**(AudioManager** audioManager**)** **{**

**if** **(**audioManager**.**getRingerMode**()** **!=** **RINGER\_MODE.**RINGER\_MODE\_SILENT**)** **{**

**int** max **=** audioManager**.**getStreamMaxVolume**();**

audioManager**.**setStreamVolume**(**max**);**

**}**

audioManager**.**setStreamVolume**(50);**

**}**

**}**

### [12.2. Testing VolumeUtil](https://www.vogella.com/tutorials/Mockito/article.html#testing-volumeutil)

We want to test VolumeUtil using Mockito.

**package** **com.vogella.mockito.audio;**

**import** **static** org**.**mockito**.**Mockito**.**mock**;**

**import** **static** org**.**mockito**.**Mockito**.**verify**;**

**import** **static** org**.**mockito**.**Mockito**.**verifyNoMoreInteractions**;**

**import** **static** org**.**mockito**.**Mockito**.**when**;**

**import** **org.junit.jupiter.api.Test;**

**class** **VolumeUtilTests** **{**

**@Test**

**void** testNormalRingerIsMaximized**(){**

*// 1.) Ensure AudioManager gets mocked*

*// 2.) configure Audiomanager to return RINGER\_MODE\_NORMAL if getRinderMode is called*

*// 3.) configure Audiomanager to return 100 if getStreamMaxVolume() is called*

*// 4.) call VolumeUtil.maximizeVolume with Audiomanager -> code under test*

*// 5.) verify that setStreamVolume(100) was called on audioManager*

**}**

**@Test**

**void** testSilentRingerIsNotDisturbed**()** **{**

*// 1.) Ensure AudioManager gets mocked*

*// 2.) configure audiomanager to return "RINGER\_MODE\_SILENT" if getRingerMode is called*

*// 3.) call VolumeUtil.maximizeVolume with audio manager*

*// 4.) verify that getRingerMode() is called on the mock*

*// 5.) Ensure that nothing more was called*

**}**

**}**

Solution

The second test should find an error in VolumeUtil. Fix this in the original code.

Solution

### [12.3. Write a new Mockito test for ConfigureThreadingUtil](https://www.vogella.com/tutorials/Mockito/article.html#write-a-new-mockito-test-for-configurethreadingutil)

Write a new test which implements the following test comments using the following template.

**package** **com.vogella.mockito.audio;**

**import** **org.junit.jupiter.api.Test;**

**class** **ConfigureThreadingUtilTests** **{**

**@Test**

**void** ensureThatThreadPoolCanBeConfigured**()** **{**

*// mock MyApplication*

*// call ConfigureThreadingUtil.configureThreadPool*

*// verify that getNumberOfThreads was the only one called on app*

**}**

**}**

Solution

## [13. Exercise: Using @Spy from Mockito](https://www.vogella.com/tutorials/Mockito/article.html#exercise-using-spy-from-mockito)

Write a test named TestingSpy. Configure a List<String> with the @Spy annotation to return a certain string for the get(10000000) on on the list.

HINT: Use doReturn to configure the spy

Show Solution

## [14. Exercise: Using @InjectMock](https://www.vogella.com/tutorials/Mockito/article.html#exercise-using-injectmock)

In this exercise you use @InjectMock to get your mocks injected into another mock.

### [14.1. Create a simple data model](https://www.vogella.com/tutorials/Mockito/article.html#create-a-simple-data-model)

Create for this example the following classes.

**package** **com.vogella.mockito.inject;**

**public** **class** **User** **{**

**String** name**;**

**public** User**(String** name**)** **{**

**this.**name **=** name**;**

**}**

**}**

**package** **com.vogella.mockito.inject;**

**public** **class** **ArticleListener** **{**

**}**

**package** **com.vogella.mockito.inject;**

**public** **class** **ArticleDatabase** **{**

**private** **ArticleListener** articleListener**;**

**private** **User** user**;**

**public** **void** addListener**(ArticleListener** articleListener**)** **{**

**this.**articleListener **=** articleListener**;**

**}**

**public** **void** setUser**(User** user**)** **{**

**this.**user **=** user**;**

**}**

**}**

**package** **com.vogella.mockito.inject;**

**import** **java.util.Objects;**

**public** **class** **ArticleManager** **{**

**private** **User** user**;**

**private** **ArticleDatabase** database**;**

*// not necessary for Mockito but a Java standard supported by many framework*

*// would be the usage of @Inject from Java Specification Request (JSR 330)*

*// @Inject*

**public** ArticleManager**(User** user**,** **ArticleDatabase** database**)** **{**

**Objects.**requireNonNull**(**user**);**

**Objects.**requireNonNull**(**database**);**

**this.**user **=** user**;**

**this.**database **=** database**;**

**}**

**public** **void** initialize**()** **{**

database**.**addListener**(new** **ArticleListener());**

database**.**setUser**(**user**);**

**}**

**}**

### [14.2. Create a test](https://www.vogella.com/tutorials/Mockito/article.html#create-a-test)

Create a new test named ArticleManagerTest uses

**package** **com.vogella.mockito.inject;**

**import** **static** org**.**mockito**.**ArgumentMatchers**.**any**;**

**import** **static** org**.**mockito**.**Mockito**.**verify**;**

**import** **org.junit.jupiter.api.Test;**

**import** **org.junit.jupiter.api.extension.ExtendWith;**

**import** **org.mockito.InjectMocks;**

**import** **org.mockito.Mock;**

**import** **org.mockito.junit.jupiter.MockitoExtension;**

**@ExtendWith(MockitoExtension.**class**)**

**class** **ArticleManagerTest** **{**

*//*

**@InjectMocks**

**private** **ArticleManager** manager**;**

**@Test**

**void** ensureInjectMockWorks**()** **{**

*// calls addListener with an instance of ArticleListener*

manager**.**initialize**();**

*// TODO verify that addListener was called with any (instance) of ArticleListener.class*

*// TODO*

**}**

**}**

If you run this test, you get an error.

org.mockito.exceptions.misusing.InjectMocksException:

Cannot instantiate @InjectMocks field named 'manager' of type 'class com.vogella.mockito.inject.ArticleManager'.

You haven't provided the instance at field declaration so I tried to construct the instance.

However the constructor or the initialization block threw an exception : null

Fix this by providing the required mocks. Also implement the TODOs.

### [14.3. Solution](https://www.vogella.com/tutorials/Mockito/article.html#solution)

Solution

## [15. Exercise: Mocking final classes and static methods](https://www.vogella.com/tutorials/Mockito/article.html#exercise-mocking-final-classes-and-static-methods)

Recent releases of Mockito can also mock final classes and static methods. For final classes and final methods Mockito just works, for static methods you have to use the mockStatic method.

**package** **com.vogella.mockito.mockstatic;**

**import** **static** org**.**junit**.**jupiter**.**api**.**Assertions**.**assertEquals**;**

**import** **static** org**.**mockito**.**ArgumentMatchers**.**anyString**;**

**import** **org.junit.jupiter.api.Test;**

**import** **org.mockito.MockedStatic;**

**import** **org.mockito.Mockito;**

**class** **MyStaticDemoTest** **{**

**@Test**

**void** testStaticMockVoid**()** **{**

**try** **(MockedStatic<Dummy>** dummy **=** **Mockito.**mockStatic**(Dummy.**class**))** **{**

dummy**.**when**(Dummy::**foo**).**thenReturn**("mocked");**

dummy**.**when**(()** **->** **Dummy.**foo**(**anyString**())).**thenReturn**("mockedValue");**

assertEquals**("mocked",** **Dummy.**foo**());**

assertEquals**("mockedValue",** **Dummy.**foo**("para"));**

dummy**.**verify**(()** **->** **Dummy.**foo**());**

dummy**.**verify**(()** **->** **Dummy.**foo**(**anyString**()));**

**}**

**}**

**static** **final** **class** **Dummy** **{**

**public** **int** testing**()** **{**

**return** var1**.**length**();**

**}**

**static** **String** var1 **=** **null;**

**static** **String** foo**()** **{**

**return** **"foo";**

**}**

**static** **String** foo**(String** var2**)** **{**

var1 **=** var2**;**

**return** **"SUCCESS";**

**}**

**}**

**}**

### [15.1. Creating a bigger example static mock test](https://www.vogella.com/tutorials/Mockito/article.html#creating-a-bigger-example-static-mock-test)

In this exercise, you create a test using to for this copied from the Mockito tests and migrated to JUnit5.

Create the following test and review it to see that is possible with static mocking in Mockito.

*/\**

*\* Copyright (c) 2020 Mockito contributors*

*\* This program is made available under the terms of the MIT License.*

*\*/*

**package** **com.vogella.mockito.mockstatic;**

**import** **static** org**.**junit**.**jupiter**.**api**.**Assertions**.**assertEquals**;**

**import** **static** org**.**junit**.**jupiter**.**api**.**Assertions**.**assertNull**;**

**import** **static** org**.**junit**.**jupiter**.**api**.**Assertions**.**assertThrows**;**

**import** **static** org**.**junit**.**jupiter**.**api**.**Assertions**.**fail**;**

**import** **static** org**.**mockito**.**Mockito**.**mockStatic**;**

**import** **static** org**.**mockito**.**Mockito**.**times**;**

**import** **java.util.concurrent.atomic.AtomicReference;**

**import** **org.junit.jupiter.api.Test;**

**import** **org.mockito.MockedStatic;**

**import** **org.mockito.exceptions.base.MockitoException;**

**import** **org.mockito.exceptions.verification.NoInteractionsWanted;**

**import** **org.mockito.exceptions.verification.WantedButNotInvoked;**

**class** **StaticMockTest** **{**

**@Test**

**void** testStaticMockSimple**()** **{**

assertEquals**("foo",** **Dummy.**foo**());**

**try** **(MockedStatic<Dummy>** ignored **=** mockStatic**(Dummy.**class**))** **{**

assertNull**(Dummy.**foo**());**

**}**

assertEquals**("foo",** **Dummy.**foo**());**

**}**

**@Test**

**void** testStaticMockWithVerification**()** **{**

**try** **(MockedStatic<Dummy>** dummy **=** mockStatic**(Dummy.**class**))** **{**

dummy**.**when**(Dummy::**foo**).**thenReturn**("bar");**

assertEquals**("bar",** **Dummy.**foo**());**

dummy**.**verify**(Dummy::**foo**);**

**}**

**}**

**void** testStaticMockWithVerificationFailed**()** **{**

**try** **(MockedStatic<Dummy>** dummy **=** mockStatic**(Dummy.**class**))** **{**

assertThrows**(WantedButNotInvoked.**class**,** **()** **->** **{**

dummy**.**verify**(Dummy::**foo**);**

**});**

**}**

**}**

**@Test**

**void** testStaticMockWithMoInteractions**()** **{**

**try** **(MockedStatic<Dummy>** dummy **=** mockStatic**(Dummy.**class**))** **{**

dummy**.**when**(Dummy::**foo**).**thenReturn**("bar");**

dummy**.**verifyNoInteractions**();**

**}**

**}**

**void** testStaticMockWithMoInteractionsFailed**()** **{**

**try** **(MockedStatic<Dummy>** dummy **=** mockStatic**(Dummy.**class**))** **{**

dummy**.**when**(Dummy::**foo**).**thenReturn**("bar");**

assertEquals**("bar",** **Dummy.**foo**());**

assertThrows**(NoInteractionsWanted.**class**,** **()** **->** **{**

dummy**.**verifyNoInteractions**();**

**});**

**}**

**}**

**@Test**

**void** testStaticMockWithMoMoreInteractions**()** **{**

**try** **(MockedStatic<Dummy>** dummy **=** mockStatic**(Dummy.**class**))** **{**

dummy**.**when**(Dummy::**foo**).**thenReturn**("bar");**

assertEquals**("bar",** **Dummy.**foo**());**

dummy**.**verify**(Dummy::**foo**);**

dummy**.**verifyNoMoreInteractions**();**

**}**

**}**

**void** testStaticMockWithMoMoreInteractionsFailed**()** **{**

**try** **(MockedStatic<Dummy>** dummy **=** mockStatic**(Dummy.**class**))** **{**

dummy**.**when**(Dummy::**foo**).**thenReturn**("bar");**

assertEquals**("bar",** **Dummy.**foo**());**

assertThrows**(NoInteractionsWanted.**class**,** **()** **->** **{**

dummy**.**verifyNoMoreInteractions**();**

**});**

**}**

**}**

**@Test**

**void** testStaticMockWithDefaultAnswer**()** **{**

**try** **(MockedStatic<Dummy>** dummy **=** mockStatic**(Dummy.**class**,** invocation **->** **"bar"))** **{**

assertEquals**("bar",** **Dummy.**foo**());**

dummy**.**verify**(Dummy::**foo**);**

**}**

**}**

**@Test**

**void** testStaticMockWithRealMethodCall**()** **{**

**try** **(MockedStatic<Dummy>** dummy **=** mockStatic**(Dummy.**class**))** **{**

dummy**.**when**(Dummy::**foo**).**thenCallRealMethod**();**

assertEquals**("foo",** **Dummy.**foo**());**

dummy**.**verify**(Dummy::**foo**);**

**}**

**}**

**@Test**

**void** testStaticMockReset**()** **{**

**try** **(MockedStatic<Dummy>** dummy **=** mockStatic**(Dummy.**class**))** **{**

dummy**.**when**(Dummy::**foo**).**thenReturn**("bar");**

dummy**.**reset**();**

assertNull**(Dummy.**foo**());**

**}**

**}**

**@Test**

**void** testStaticMockClear**()** **{**

**try** **(MockedStatic<Dummy>** dummy **=** mockStatic**(Dummy.**class**))** **{**

dummy**.**when**(Dummy::**foo**).**thenReturn**("bar");**

assertEquals**("bar",** **Dummy.**foo**());**

dummy**.**clearInvocations**();**

dummy**.**verifyNoInteractions**();**

**}**

**}**

**@Test**

**void** testStaticMockDoesNotAffectDifferentThread**()** **throws** **InterruptedException** **{**

**try** **(MockedStatic<Dummy>** dummy **=** mockStatic**(Dummy.**class**))** **{**

dummy**.**when**(Dummy::**foo**).**thenReturn**("bar");**

assertEquals**("bar",** **Dummy.**foo**());**

dummy**.**verify**(Dummy::**foo**);**

**AtomicReference<String>** reference **=** **new** **AtomicReference<>();**

**Thread** thread **=** **new** **Thread(()** **->** reference**.**set**(Dummy.**foo**()));**

thread**.**start**();**

thread**.**join**();**

assertEquals**("foo",** reference**.**get**());**

dummy**.**when**(Dummy::**foo**).**thenReturn**("bar");**

assertEquals**("bar",** **Dummy.**foo**());**

dummy**.**verify**(**times**(2),** **Dummy::**foo**);**

**}**

**}**

**@Test**

**void** testStaticMockCanCoexistWithMockInDifferentThread**()** **throws** **InterruptedException** **{**

**try** **(MockedStatic<Dummy>** dummy **=** mockStatic**(Dummy.**class**))** **{**

dummy**.**when**(Dummy::**foo**).**thenReturn**("bar");**

assertEquals**("bar",** **Dummy.**foo**());**

dummy**.**verify**(Dummy::**foo**);**

**AtomicReference<String>** reference **=** **new** **AtomicReference<>();**

**Thread** thread **=** **new** **Thread(()** **->** **{**

**try** **(MockedStatic<Dummy>** dummy2 **=** mockStatic**(Dummy.**class**))** **{**

dummy2**.**when**(Dummy::**foo**).**thenReturn**("qux");**

reference**.**set**(Dummy.**foo**());**

**}**

**});**

thread**.**start**();**

thread**.**join**();**

assertEquals**("qux",** reference**.**get**());**

dummy**.**when**(Dummy::**foo**).**thenReturn**("bar");**

assertEquals**("bar",** **Dummy.**foo**());**

dummy**.**verify**(**times**(2),** **Dummy::**foo**);**

**}**

**}**

**void** testStaticMockMustBeExclusiveInScopeWithinThread**()** **{**

**try** **{**

**try** **(MockedStatic<Dummy>** dummy **=** mockStatic**(Dummy.**class**);**

**MockedStatic<Dummy>** duplicate **=** mockStatic**(Dummy.**class**))** **{**

fail**("Not supposed to allow duplicates");**

**}**

**}** **catch** **(Exception** e**)** **{**

assertEquals**(MockitoException.**class**,** e**.**getClass**());**

**}**

**}**

**@Test**

**void** testStaticMockVoid**()** **{**

**try** **(MockedStatic<Dummy>** dummy **=** mockStatic**(Dummy.**class**))** **{**

**Dummy.**fooVoid**("bar");**

assertNull**(Dummy.**var1**);**

dummy**.**verify**(()** **->** **Dummy.**fooVoid**("bar"));**

**}**

**Dummy.**fooVoid**("bar");**

assertEquals**("bar",** **Dummy.**var1**);**

**}**

**static** **class** **Dummy** **{**

**static** **String** var1 **=** **null;**

**static** **String** foo**()** **{**

**return** **"foo";**

**}**

**static** **void** fooVoid**(String** var2**)** **{**

var1 **=** var2**;**

**}**

**}**

**}**

## [16. Exercise: Using Spy and reflection to change private fields](https://www.vogella.com/tutorials/Mockito/article.html#exercise-using-spy-and-reflection-to-change-private-fields)

Mockito can currently not mock private fields or methods. Changing private fields or methods during test should be avoided you you should try to refactor you code.

Technically you could use a combination with @Spy and Java reflection access. This exercise demonstrates that.

### [16.1. Create data model](https://www.vogella.com/tutorials/Mockito/article.html#create-data-model)

Create the following class.

**package** **com.vogella.mockito.withprivate;**

**public** **class** **MyClassWithPrivateFieldAndMethod** **{**

**public** **String** field1 **=** **"";**

**public** **String** valueSetByPrivateMethod **=** **"";**

**private** **String** hiddenField **=** **"initial";**

**public** **String** getValue**()** **{**

**return** hiddenField**;**

**}**

**public** **String** getValueSetByPrivateMethod**()** **{**

**return** valueSetByPrivateMethod**;**

**}**

**public** **String** toBeMockedByMockito**()** **{**

**return** **"stuff";**

**}**

**private** **void** meineMethod**()** **{**

valueSetByPrivateMethod **=** **"lalal";**

**}**

**}**

### [16.2. Use Spy and reflection](https://www.vogella.com/tutorials/Mockito/article.html#use-spy-and-reflection)

Now assume you want to write a test and mock the toBeMockedByMockito method.

**package** **com.vogella.mockito;**

**import** **static** org**.**junit**.**jupiter**.**api**.**Assertions**.**assertEquals**;**

**import** **static** org**.**mockito**.**Mockito**.**when**;**

**import** **java.lang.reflect.Field;**

**import** **java.lang.reflect.InvocationTargetException;**

**import** **java.lang.reflect.Method;**

**import** **org.junit.jupiter.api.Test;**

**import** **org.junit.jupiter.api.extension.ExtendWith;**

**import** **org.mockito.Spy;**

**import** **org.mockito.junit.jupiter.MockitoExtension;**

**import** **com.vogella.mockito.withprivate.MyClassWithPrivateFieldAndMethod;**

**@ExtendWith(MockitoExtension.**class**)**

**class** **MyClassWithPrivateFieldAndMethodTest** **{**

**@Spy**

**MyClassWithPrivateFieldAndMethod** mock **=** **new** **MyClassWithPrivateFieldAndMethod();**

**@Test**

**void** ensureSpyAndReflectiveAccessCanChangeAPrivateField**()** **throws** **NoSuchFieldException,** **SecurityException,**

**IllegalArgumentException,** **IllegalAccessException,** **NoSuchMethodException,** **InvocationTargetException** **{**

assertEquals**("initial",** mock**.**getValue**());**

mock**.**field1 **=** **"Hello";**

when**(**mock**.**toBeMockedByMockito**()).**thenReturn**("mocked by Mockito");**

**Field** declaredField **=** **MyClassWithPrivateFieldAndMethod.**class**.**getDeclaredField**("hiddenField");**

declaredField**.**setAccessible**(true);**

declaredField**.**set**(**mock**,** **"changed");**

assertEquals**("Hello",** mock**.**field1**);**

assertEquals**("changed",** mock**.**getValue**());**

assertEquals**("mocked by Mockito",** mock**.**toBeMockedByMockito**());**

**}**

**}**

how to test the Service layer in the Spring Boot application using **JUnit 5** and **Mockito**framework.

**[](https://lh3.googleusercontent.com/-M9j0Ir8_9wE/YShvmyTMCAI/AAAAAAAAJCE/riHVO6MDks4yRL4i4uEkCtBSq3IPb3IjwCLcBGAsYHQ/image.png)**

# Create Spring Boot Project

We’ll use [**Spring initializr**](https://start.spring.io/) web tool to bootstrap our application.

Go to [**http://start.spring.io**](http://start.spring.io/)

Select **Java**in the language section.

Enter Artifact as **springboot-testing-demo**

Add **Lombok**, **JPA**, and **MySQL** dependencies.

Click Generate to generate and download the project.

Once the project is generated, unzip it and import it into your favorite IDE.

# Maven Dependencies

Here is a complete pom.xml file for your reference:

<?xml version="1.0" encoding="UTF-8"?>

<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<parent>

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

<artifactId>spring-boot-starter-parent</artifactId>

<version>2.4.5</version>

<relativePath/> <!-- lookup parent from repository -->

</parent>

<groupId>com.example</groupId>

<artifactId>Spring-boot-tutorial</artifactId>

<version>1.0.0</version>

<name>Spring-boot-tutorial</name>

<description>Demo project for Spring Boot</description>

<properties>

<java.version>11</java.version>

</properties>

<dependencies>

<dependency>

<groupId>mysql</groupId>

<artifactId>mysql-connector-java</artifactId>

<scope>runtime</scope>

</dependency>

<dependency>

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

<artifactId>spring-boot-starter-data-jpa</artifactId>

</dependency>

<dependency>

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

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

<scope>test</scope>

</dependency>

<dependency>

<groupId>org.projectlombok</groupId>

<artifactId>lombok</artifactId>

<optional>true</optional>

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

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

<artifactId>spring-boot-maven-plugin</artifactId>

<configuration>

<excludes>

<exclude>

<groupId>org.projectlombok</groupId>

<artifactId>lombok</artifactId>

</exclude>

</excludes>

</configuration>

</plugin>

</plugins>

</build>

</project>

We have added dependencies:

* spring-boot-starter-test - main dependencies for unit and integration testing.
* spring-boot-starter-data-jpa - to connect to a database and perform DB operations
* lombok - to reduce boilerplate code

The *spring-boot-maven-plugin* is used to create an executable jar with the Spring Boot application.

# Configure MySQL Database

We’ll need to configure MySQL database URL, username, and password so that Spring Boot can create a Data source.

Open src/main/resources/application.properties file and add the following properties to it -

spring.datasource.url = jdbc:mysql://localhost:3306/demo?autoReconnect=true&useUnicode=true&characterEncoding=UTF-8&allowMultiQueries=true&useSSL=false

spring.datasource.username = root

spring.datasource.password = root

spring.jpa.properties.hibernate.dialect = org.hibernate.dialect.MySQL5InnoDBDialect

spring.jpa.hibernate.ddl-auto = update

Change spring.datasource.username and spring.datasource.password as per your MySQL installation.

Note that, I’ve set spring.jpa.hibernate.ddl-auto property to update. This property updates the database schema whenever you create or modify the domain models in your application.

# Create JPA Entity

Let's create a *Department* JPA entity:

import lombok.\*;

import javax.persistence.Entity;

import javax.persistence.GeneratedValue;

import javax.persistence.GenerationType;

import javax.persistence.Id;

import javax.validation.constraints.NotBlank;

@Entity

@Data

@NoArgsConstructor

@AllArgsConstructor

@Builder

public class Department {

@Id

@GeneratedValue(strategy = GenerationType.AUTO)

private Long departmentId;

private String departmentName;

private String departmentAddress;

private String departmentCode;

}

# Create Spring Data JPA Repository

Let’s now create the repository for accessing the data from the database. Let's create a *DepartmentRepository* interface and extend it to *JpaRepository* to get CRUD methods:

import com.example.Springboot.tutorial.entity.Department;

import org.springframework.data.jpa.repository.JpaRepository;

import org.springframework.stereotype.Repository;

@Repository

public interface DepartmentRepository extends JpaRepository<Department, Long> {

public Department findByDepartmentName(String departmentName);

public Department findByDepartmentNameIgnoreCase(String departmentName);

}

# Service Layer

To keep it simple, we are going to write a JUnit test case for a single method *fetchDepartmentByName()*:

## DepartmentService

import com.example.Springboot.tutorial.entity.Department;

import java.util.List;

public interface DepartmentService {

Department fetchDepartmentByName(String departmentName);

}

## DepartmentServiceImpl

import com.example.Springboot.tutorial.entity.Department;

import com.example.Springboot.tutorial.repository.DepartmentRepository;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Service;

import java.util.List;

import java.util.Objects;

import java.util.Optional;

@Service

public class DepartmentServiceImpl implements DepartmentService{

@Autowired

private DepartmentRepository departmentRepository;

@Override

public Department fetchDepartmentByName(String departmentName) {

return departmentRepository.findByDepartmentNameIgnoreCase(departmentName);

}

}

We have implemented the server layer, next we will write a simple JUnit test for DepartmentService.

# Testing Service Layer

Let's go ahead and use JUnit 5 and Mockito to write a JUnit test case.

Note that instead of connecting to a database, we are mocking the objects and testing only the Service layer.

Let's create a *DepartmentServiceTest* class and annotate it with *@SpringBootTest* annotation like this:

import com.example.Springboot.tutorial.entity.Department;

import com.example.Springboot.tutorial.repository.DepartmentRepository;

import org.junit.jupiter.api.BeforeEach;

import org.junit.jupiter.api.DisplayName;

import org.junit.jupiter.api.Test;

import org.mockito.Mockito;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.boot.test.context.SpringBootTest;

import org.springframework.boot.test.mock.mockito.MockBean;

import static org.junit.jupiter.api.Assertions.assertEquals;

@SpringBootTest

class DepartmentServiceTest {

@Autowired

private DepartmentService departmentService;

@MockBean

private DepartmentRepository departmentRepository;

@BeforeEach

void setUp() {

Department department =

Department.builder()

.departmentName("IT")

.departmentAddress("Ahmedabad")

.departmentCode("IT-06")

.departmentId(1L)

.build();

Mockito.when(departmentRepository.findByDepartmentNameIgnoreCase("IT"))

.thenReturn(department);

}

@Test

@DisplayName("Get Data based on Valida Department Name")

public void whenValidDepartmentName\_thenDepartmentShouldFound() {

String departmentName = "IT";

Department found =

departmentService.fetchDepartmentByName(departmentName);

assertEquals(departmentName, found.getDepartmentName());

}

}

We are mocking the *DepartmentRepository*interface using *@MockBean* annotation:

@MockBean

private DepartmentRepository departmentRepository;

The *@SpringBootTest* annotation loads the complete Spring application context. In contrast, a test slice annotation only loads beans required to test a particular layer. And because of this, we can avoid unnecessary mocking and side effects.

We can use the *@MockBean* to add mock objects to the Spring application context. The mock will replace any existing bean of the same type in the application context.

*@BeforeEach* is used to signal that the annotated method should be executed before each *@Test* method in the current class:

@BeforeEach

void setUp() {

Department department =

Department.builder()

.departmentName("IT")

.departmentAddress("Ahmedabad")

.departmentCode("IT-06")

.departmentId(1L)

.build();

Mockito.when(departmentRepository.findByDepartmentNameIgnoreCase("IT"))

.thenReturn(department);

}

We wrote a JUnit test for *fetchDepartmentByName()* method:

@Test

@DisplayName("Get Data based on Valida Department Name")

public void whenValidDepartmentName\_thenDepartmentShouldFound() {

String departmentName = "IT";

Department found =

departmentService.fetchDepartmentByName(departmentName);

assertEquals(departmentName, found.getDepartmentName());

}

Similarly, you can write JUnit test cases for all the positive and negative scenarios.

Example2:

* Spring Boot 2.1.2.RELEASE
* JUnit 5
* Mockito 2
* Maven 3

In short, exclude junit4 from spring-boot-starter-test, and include the JUnit 5 jupiter engine manually, done.

Let see the following Spring boot MVC web application, and how to perform unit test with JUnit 5 and mocking with Mockito framework.

## **1. Maven**

pom.xml

<?xml version="1.0" encoding="UTF-8"?>

<project xmlns="http://maven.apache.org/POM/4.0.0"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0

http://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>com.mkyong.spring</groupId>

<artifactId>testing-junit5-mockito</artifactId>

<version>1.0</version>

<properties>

<java.version>1.8</java.version>

<junit-jupiter.version>5.3.2</junit-jupiter.version>

<mockito.version>2.24.0</mockito.version>

</properties>

<parent>

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

<artifactId>spring-boot-starter-parent</artifactId>

<version>2.1.2.RELEASE</version>

</parent>

<dependencies>

*<!-- mvc -->*

<dependency>

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

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

*<!-- exclude junit 4 -->*

<dependency>

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

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

<scope>test</scope>

<exclusions>

<exclusion>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

</exclusion>

</exclusions>

</dependency>

*<!-- junit 5 -->*

<dependency>

<groupId>org.junit.jupiter</groupId>

<artifactId>junit-jupiter-engine</artifactId>

<version>${junit-jupiter.version}</version>

<scope>test</scope>

</dependency>

<dependency>

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

<artifactId>spring-boot-devtools</artifactId>

<optional>true</optional>

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

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

<artifactId>spring-boot-maven-plugin</artifactId>

</plugin>

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-surefire-plugin</artifactId>

<version>2.22.0</version>

</plugin>

</plugins>

</build>

</project>

Display the project dependencies.

$ mvn dependency:tree

[INFO] com.mkyong.spring:testing-junit5-mockito:jar:1.0

[INFO] +- org.springframework.boot:spring-boot-starter-web:jar:2.1.2.RELEASE:compile

[INFO] | +- org.springframework.boot:spring-boot-starter:jar:2.1.2.RELEASE:compile

[INFO] | | +- org.springframework.boot:spring-boot-starter-logging:jar:2.1.2.RELEASE:compile

[INFO] | | | +- ch.qos.logback:logback-classic:jar:1.2.3:compile

[INFO] | | | | \- ch.qos.logback:logback-core:jar:1.2.3:compile

[INFO] | | | +- org.apache.logging.log4j:log4j-to-slf4j:jar:2.11.1:compile

[INFO] | | | | \- org.apache.logging.log4j:log4j-api:jar:2.11.1:compile

[INFO] | | | \- org.slf4j:jul-to-slf4j:jar:1.7.25:compile

[INFO] | | +- javax.annotation:javax.annotation-api:jar:1.3.2:compile

[INFO] | | \- org.yaml:snakeyaml:jar:1.23:runtime

[INFO] | +- org.springframework.boot:spring-boot-starter-json:jar:2.1.2.RELEASE:compile

[INFO] | | +- com.fasterxml.jackson.core:jackson-databind:jar:2.9.8:compile

[INFO] | | | +- com.fasterxml.jackson.core:jackson-annotations:jar:2.9.0:compile

[INFO] | | | \- com.fasterxml.jackson.core:jackson-core:jar:2.9.8:compile

[INFO] | | +- com.fasterxml.jackson.datatype:jackson-datatype-jdk8:jar:2.9.8:compile

[INFO] | | +- com.fasterxml.jackson.datatype:jackson-datatype-jsr310:jar:2.9.8:compile

[INFO] | | \- com.fasterxml.jackson.module:jackson-module-parameter-names:jar:2.9.8:compile

[INFO] | +- org.springframework.boot:spring-boot-starter-tomcat:jar:2.1.2.RELEASE:compile

[INFO] | | +- org.apache.tomcat.embed:tomcat-embed-core:jar:9.0.14:compile

[INFO] | | +- org.apache.tomcat.embed:tomcat-embed-el:jar:9.0.14:compile

[INFO] | | \- org.apache.tomcat.embed:tomcat-embed-websocket:jar:9.0.14:compile

[INFO] | +- org.hibernate.validator:hibernate-validator:jar:6.0.14.Final:compile

[INFO] | | +- javax.validation:validation-api:jar:2.0.1.Final:compile

[INFO] | | +- org.jboss.logging:jboss-logging:jar:3.3.2.Final:compile

[INFO] | | \- com.fasterxml:classmate:jar:1.4.0:compile

[INFO] | +- org.springframework:spring-web:jar:5.1.4.RELEASE:compile

[INFO] | | \- org.springframework:spring-beans:jar:5.1.4.RELEASE:compile

[INFO] | \- org.springframework:spring-webmvc:jar:5.1.4.RELEASE:compile

[INFO] | +- org.springframework:spring-aop:jar:5.1.4.RELEASE:compile

[INFO] | +- org.springframework:spring-context:jar:5.1.4.RELEASE:compile

[INFO] | \- org.springframework:spring-expression:jar:5.1.4.RELEASE:compile

[INFO] +- org.springframework.boot:spring-boot-starter-test:jar:2.1.2.RELEASE:test

[INFO] | +- org.springframework.boot:spring-boot-test:jar:2.1.2.RELEASE:test

[INFO] | +- org.springframework.boot:spring-boot-test-autoconfigure:jar:2.1.2.RELEASE:test

[INFO] | +- com.jayway.jsonpath:json-path:jar:2.4.0:test

[INFO] | | +- net.minidev:json-smart:jar:2.3:test

[INFO] | | | \- net.minidev:accessors-smart:jar:1.2:test

[INFO] | | | \- org.ow2.asm:asm:jar:5.0.4:test

[INFO] | | \- org.slf4j:slf4j-api:jar:1.7.25:compile

[INFO] | +- org.assertj:assertj-core:jar:3.11.1:test

[INFO] | +- org.mockito:mockito-core:jar:2.24.0:test

[INFO] | | +- net.bytebuddy:byte-buddy:jar:1.9.7:test

[INFO] | | +- net.bytebuddy:byte-buddy-agent:jar:1.9.7:test

[INFO] | | \- org.objenesis:objenesis:jar:2.6:test

[INFO] | +- org.hamcrest:hamcrest-core:jar:1.3:test

[INFO] | +- org.hamcrest:hamcrest-library:jar:1.3:test

[INFO] | +- org.skyscreamer:jsonassert:jar:1.5.0:test

[INFO] | | \- com.vaadin.external.google:android-json:jar:0.0.20131108.vaadin1:test

[INFO] | +- org.springframework:spring-core:jar:5.1.4.RELEASE:compile

[INFO] | | \- org.springframework:spring-jcl:jar:5.1.4.RELEASE:compile

[INFO] | +- org.springframework:spring-test:jar:5.1.4.RELEASE:test

[INFO] | \- org.xmlunit:xmlunit-core:jar:2.6.2:test

[INFO] | \- javax.xml.bind:jaxb-api:jar:2.3.1:test

[INFO] | \- javax.activation:javax.activation-api:jar:1.2.0:test

[INFO] +- org.junit.jupiter:junit-jupiter-engine:jar:5.3.2:test

[INFO] | +- org.apiguardian:apiguardian-api:jar:1.0.0:test

[INFO] | +- org.junit.platform:junit-platform-engine:jar:1.3.2:test

[INFO] | | +- org.junit.platform:junit-platform-commons:jar:1.3.2:test

[INFO] | | \- org.opentest4j:opentest4j:jar:1.1.1:test

[INFO] | \- org.junit.jupiter:junit-jupiter-api:jar:5.3.2:test

[INFO] \- org.springframework.boot:spring-boot-devtools:jar:2.1.2.RELEASE:compile (optional)

[INFO] +- org.springframework.boot:spring-boot:jar:2.1.2.RELEASE:compile

[INFO] \- org.springframework.boot:spring-boot-autoconfigure:jar:2.1.2.RELEASE:compile

## **2. Test Spring Boot + JUnit 5 + Mockito.**

2.1 Test the following Spring components.

HelloServiceImpl.java

package com.mkyong.core.services;

import com.mkyong.core.repository.HelloRepository;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Service;

@Service

public class HelloServiceImpl implements HelloService {

@Autowired

HelloRepository helloRepository;

@Override

public String get() {

return helloRepository.get();

}

}

HelloRepositoryImpl.java

package com.mkyong.core.repository;

import org.springframework.stereotype.Repository;

@Repository

public class HelloRepositoryImpl implements HelloRepository {

@Override

public String get() {

return "Hello JUnit 5";

}

}

2.2 JUnit 5

HelloServiceTest.java

package com.mkyong.core.services;

import org.junit.jupiter.api.DisplayName;

import org.junit.jupiter.api.Test;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.boot.test.context.SpringBootTest;

import static org.junit.jupiter.api.Assertions.assertEquals;

@SpringBootTest

public class HelloServiceTest {

@Autowired

HelloService helloService;

@DisplayName("Test Spring @Autowired Integration")

@Test

void testGet() {

assertEquals("Hello JUnit 5", helloService.get());

}

}

2.3 Mockito.

HelloServiceMockTest.java

package com.mkyong.core.services;

import com.mkyong.core.repository.HelloRepository;

import org.junit.jupiter.api.BeforeEach;

import org.junit.jupiter.api.DisplayName;

import org.junit.jupiter.api.Test;

import org.mockito.InjectMocks;

import org.mockito.Mock;

import org.springframework.boot.test.context.SpringBootTest;

import static org.junit.jupiter.api.Assertions.assertEquals;

import static org.mockito.Mockito.when;

@SpringBootTest

public class HelloServiceMockTest {

@Mock

private HelloRepository helloRepository;

@InjectMocks *// auto inject helloRepository*

private HelloService helloService = new HelloServiceImpl();

@BeforeEach

void setMockOutput() {

when(helloRepository.get()).thenReturn("Hello Mockito From Repository");

}

@DisplayName("Test Mock helloService + helloRepository")

@Test

void testGet() {

assertEquals("Hello Mockito From Repository", helloService.get());

}

}

## **3. Test Spring MVC Controller.**

3.1 MVC controller.

MainController.java

package com.mkyong.core.controller;

import org.springframework.stereotype.Controller;

import org.springframework.web.bind.annotation.GetMapping;

import org.springframework.web.bind.annotation.ResponseBody;

@Controller

public class MainController {

@ResponseBody

@GetMapping("/")

public String hello() {

return "Hello Controller";

}

}

3.2 JUnit 5 and MVC test.

MainControllerTest.java

package com.mkyong.core.controller;

import org.junit.jupiter.api.Test;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.boot.test.context.SpringBootTest;

import org.springframework.boot.test.context.SpringBootTest.WebEnvironment;

import org.springframework.boot.test.web.client.TestRestTemplate;

import org.springframework.boot.web.server.LocalServerPort;

import org.springframework.http.ResponseEntity;

import java.net.URL;

import static org.junit.jupiter.api.Assertions.assertEquals;

@SpringBootTest(webEnvironment = WebEnvironment.RANDOM\_PORT)

public class MainControllerTest {

*// bind the above RANDOM\_PORT*

@LocalServerPort

private int port;

@Autowired

private TestRestTemplate restTemplate;

@Test

public void getHello() throws Exception {

ResponseEntity<String> response = restTemplate.getForEntity(

new URL("http://localhost:" + port + "/").toString(), String.class);

assertEquals("Hello Controller", response.getBody());

}

}

Done.