Оглавление

[1. MATERIALS 2](#_Toc25324373)

[2. OVERVIEW 2](#_Toc25324374)

[3. FEATURES 3](#_Toc25324375)

[4. NOTES: 4](#_Toc25324376)

# MATERIALS

* <https://junit.org/junit5/docs/current/user-guide/#overview>

# OVERVIEW

JUnit Jupiter supports the following annotations for configuring tests and extending the framework.

| **Annotation** | **Description** |
| --- | --- |
| @Test | Denotes that a method is a test method. Unlike JUnit 4’s @Test annotation, this annotation does not declare any attributes, since test extensions in JUnit Jupiter operate based on their own dedicated annotations. Such methods are inherited unless they are overridden. |
| @ParameterizedTest | Denotes that a method is a [parameterized test](https://junit.org/junit5/docs/current/user-guide/#writing-tests-parameterized-tests). Such methods are inherited unless they are overridden. |
| @RepeatedTest | Denotes that a method is a test template for a [repeated test](https://junit.org/junit5/docs/current/user-guide/#writing-tests-repeated-tests). Such methods are inherited unless they are overridden. |
| @TestFactory | Denotes that a method is a test factory for [dynamic tests](https://junit.org/junit5/docs/current/user-guide/#writing-tests-dynamic-tests). Such methods are inherited unless they are overridden. |
| @TestTemplate | Denotes that a method is a [template for test cases](https://junit.org/junit5/docs/current/user-guide/#writing-tests-test-templates) designed to be invoked multiple times depending on the number of invocation contexts returned by the registered [providers](https://junit.org/junit5/docs/current/user-guide/#extensions-test-templates). Such methods are inherited unless they are overridden. |
| @TestMethodOrder | Used to configure the [test method execution order](https://junit.org/junit5/docs/current/user-guide/#writing-tests-test-execution-order) for the annotated test class; similar to JUnit 4’s @FixMethodOrder. Such annotations are inherited. |
| @TestInstance | Used to configure the [test instance lifecycle](https://junit.org/junit5/docs/current/user-guide/#writing-tests-test-instance-lifecycle) for the annotated test class. Such annotations are inherited. |
| @DisplayName | Declares a custom [display name](https://junit.org/junit5/docs/current/user-guide/#writing-tests-display-names) for the test class or test method. Such annotations are not inherited. |
| @DisplayNameGeneration | Declares a custom [display name generator](https://junit.org/junit5/docs/current/user-guide/#writing-tests-display-name-generator) for the test class. Such annotations are inherited. |
| @BeforeEach | Denotes that the annotated method should be executed before **each** @Test, @RepeatedTest, @ParameterizedTest, or @TestFactory method in the current class; analogous to JUnit 4’s @Before. Such methods are inherited unless they are overridden. |
| @AfterEach | Denotes that the annotated method should be executed after **each** @Test, @RepeatedTest, @ParameterizedTest, or @TestFactory method in the current class; analogous to JUnit 4’s @After. Such methods are inherited unless they are overridden. |
| @BeforeAll | Denotes that the annotated method should be executed before **all** @Test, @RepeatedTest, @ParameterizedTest, and @TestFactory methods in the current class; analogous to JUnit 4’s @BeforeClass. Such methods are inherited (unless they are hidden or overridden) and must be static (unless the "per-class" [test instance lifecycle](https://junit.org/junit5/docs/current/user-guide/#writing-tests-test-instance-lifecycle) is used). |
| @AfterAll | Denotes that the annotated method should be executed after **all** @Test, @RepeatedTest, @ParameterizedTest, and @TestFactory methods in the current class; analogous to JUnit 4’s @AfterClass. Such methods are inherited (unless they are hidden or overridden) and must be static (unless the "per-class" [test instance lifecycle](https://junit.org/junit5/docs/current/user-guide/#writing-tests-test-instance-lifecycle) is used). |
| @Nested | Denotes that the annotated class is a non-static [nested test class](https://junit.org/junit5/docs/current/user-guide/#writing-tests-nested). @BeforeAll and @AfterAll methods cannot be used directly in a @Nested test class unless the "per-class" [test instance lifecycle](https://junit.org/junit5/docs/current/user-guide/#writing-tests-test-instance-lifecycle) is used. Such annotations are not inherited. |
| @Tag | Used to declare [tags for filtering tests](https://junit.org/junit5/docs/current/user-guide/#writing-tests-tagging-and-filtering), either at the class or method level; analogous to test groups in TestNG or Categories in JUnit 4. Such annotations are inherited at the class level but not at the method level. |
| @Disabled | Used to [disable](https://junit.org/junit5/docs/current/user-guide/#writing-tests-disabling) a test class or test method; analogous to JUnit 4’s @Ignore. Such annotations are not inherited. |
| @Timeout | Used to fail a test, test factory, test template, or lifecycle method if its execution exceeds a given duration. Such annotations are inherited. |
| @ExtendWith | Used to [register extensions declaratively](https://junit.org/junit5/docs/current/user-guide/#extensions-registration-declarative). Such annotations are inherited. |
| @RegisterExtension | Used to [register extensions programmatically](https://junit.org/junit5/docs/current/user-guide/#extensions-registration-programmatic) via fields. Such fields are inherited unless they are shadowed. |
| @TempDir | Used to supply a [temporary directory](https://junit.org/junit5/docs/current/user-guide/#writing-tests-built-in-extensions-TempDirectory) via field injection or parameter injection in a lifecycle method or test method; located in the org.junit.jupiter.api.io package. |

**Test Class is**:

* Any top-level class.
* Static member class.
* [@Nested class](https://junit.org/junit5/docs/current/user-guide/#writing-tests-nested).

that contains at least one test method. Test classes *must not be abstract* and *must have a single constructor*.

**Test Method**: any instance method that is directly annotated or meta-annotated with:

@Test.

@RepeatedTest.

@ParameterizedTest.

@TestFactory.

@TestTemplate.

**Lifecycle Method**: any method that is directly annotated or meta-annotated with

@BeforeAll.

@AfterAll.

@BeforeEach.

@AfterEach.

**Test methods** and **lifecycle methods** may be:

* Declared locally within the current test class;
* Inherited from superclasses;
* Inherited from interfaces.

In addition, test methods and lifecycle methods *must not be abstract* and *must not return a value*. Test classes, test methods, and lifecycle methods *are not required to be public*, but they *must not be private*.

**A standard test class**:

class StandardTests {

@*BeforeAll*

static void initAll() {

}

@*BeforeEach*

void init() {

}

@*Test*

void succeedingTest() {

}

@*Test*

void failingTest() {

fail("a failing test");

}

@*Test*

@*Disabled("for demonstration purposes")*

void skippedTest() {

// not executed

}

@*Test*

void abortedTest() {

assumeTrue("abc".contains("Z"));

fail("test should have been aborted");

}

@*AfterEach*

void tearDown() {

}

@*AfterAll*

static void tearDownAll() {

}

}

# FEATURES

1. Использование мета-аннотаций и составных аннотаций, которые позволяют заменить шаблонный набор аннотаций на одну, которая будет их содержать.

@Target(ElementType.METHOD)

@Retention(RetentionPolicy.RUNTIME)

@Tag("fast")

@Test

public @interface **FastTest** {

}

**@FastTest**

void myFastTest() {

...

}

1. Настройка отображения выходной информации: имен тестовых классов и методов, результатов выполнения отдельных утверждений (assertions). Для задания имен тестов и методов может использоваться готовая аннотация **@DisplayName**, либо же создан свой генератор имен. Можно задавать генератор имен по умолчанию. Приоритет для определения того, какие имена классов или методов будут отображаться:
2. Значение параметра аннотации **@DisplayName**, если она представлена.
3. Значение, предоставляемое реализацией интерфейса **DisplayNameGenerator** через указание ее имени в параметре аннотации **@DisplayNameGeneration**, если такая аннотация представлена.
4. Значение, предоставляемое реализацией интерфейса **DisplayNameGenerator,** которая была установлена через конфигурационный параметр, если производилась конфигурация.
5. Значение, предоставляемое стандартным генератором имен - **org.junit.jupiter.api.DisplayNameGenerator.Standard**.
6. Создание методов жизненного цикла:

@BeforeAll – выполняется перед всеми тестами.

@AfterAll – выполняется после всех тестов.

@BeforeEach – выполняется перед каждым тестом.

@AfterEach – выполняется после каждого теста.

1. Использование **Assertions** – утверждений для проверки функций тестируемых классов. С помощью утверждений можно проверить:

* Равенство двух примитивных значений.
* Равенство двух массивов.
* Равенство двух Iterable объектов.
* Является ли выражение истинным или ложным.
* Равен ли аргумент null.
* Ссылаются ли обе переменных на один и тот же объект.
* Группу утверждений (Проверяет все утверждения и только потом выводит результат).
* Выброс исключения.
* Превышение временного интервала (имеются два различных метода – один запускается в том же потоке, в которой исполняется код, вызывающий тестовый метод + при превышении таймаута выполнение теста не прерывается, второй запускается в собственном потоке и прерывание выполнения происходит по таймауту).

В качестве **Assertions** можно использовать методы сторонних библиотек. Кроме того можно “завалить тест” искусственно с помощью метода **“fail”**.

1. **Assumptions** - выполнение тестов при определенных допущениях, иначе тест отменяется (“aborted”). Для этих целей используются статические методы класса *org.junit.jupiter.api.Assumptions*, например, *assumeTrue()*.
2. Исключение тестового класса или отдельных его тестовых методов из процесса тестирования (Disabling). Достигается через:

* Ааннотацию [**@Disabled**](https://junit.org/junit5/docs/current/api/org/junit/jupiter/api/Disabled.html);
* Аннотации условного выполнения тестов ([**Conditional Test Execution**](https://junit.org/junit5/docs/current/user-guide/#writing-tests-conditional-execution));
* **Custom** [**ExecutionCondition**](https://junit.org/junit5/docs/current/user-guide/#extensions-conditions).

1. Использование тегов для тестов и возможность запуска тестов по ним (Только тесты с определенными тегами будут запущены, или наоборот проигнорированы).
2. Задание порядка выполнения тестов (Можно реализовывать свои классы или использовать готовые (очередь по наименованию, номеру, случайный порядок)).
3. Изменение жизненного цикла выполнения тестового класса. (Можно создать как один общий Instance тестового класса для всех методов, так и отдельный для каждого из тестовых методов).
4. Создание вложенных тестовых классов (Nested).
5. Внедрение зависимостей в конструкторы класса и тестовые методы (С помощью классов реализующих интерфейс ParameterResolver). Три встроенные автоматически зарегистрированные реализации:

* [TestInfoParameterResolver](https://github.com/junit-team/junit5/tree/r5.5.1/junit-jupiter-engine/src/main/java/org/junit/jupiter/engine/extension/TestInfoParameterResolver.java);
* [RepetitionInfoParameterResolver](https://github.com/junit-team/junit5/tree/r5.5.1/junit-jupiter-engine/src/main/java/org/junit/jupiter/engine/extension/RepetitionInfoParameterResolver.java);
* [TestReporterParameterResolver](https://github.com/junit-team/junit5/tree/r5.5.1/junit-jupiter-engine/src/main/java/org/junit/jupiter/engine/extension/TestReporterParameterResolver.java).

Регистрация собственных классов через аннотацию @ExtendWith(\*.class)

1. Создание тестовых классов на базе интерфейсов с реализованными default методами (тестовыми методами и методами жизненного цикла тестового класса).
2. Создание тестов, которые выполняются заданное количество раз (повторяемые тесты).
3. Возможность создавать параметризированные тесты. \*Поддержка расширения типов аргументов(параметров).

# NOTES:

При использовании Maven имена тестовых классов должны соответствовать маскам:

1. Test\*
2. \*Test
3. \*TestCase

или же соответствующий плагин (maven-surefire-plugin) должен быть перенастроен.

С помощью плагина “ maven-surefire-report-plugin” можно сформировать отчет по итогам тестирования.