JUnit 5 - Jupiter

Содержание

[**0. Введение**](https://habr.com/ru/post/337700/#intro)  
[**1. Начало работы**](https://habr.com/ru/post/337700/#getting-started)  
[**2. Обзор нововведений**](https://habr.com/ru/post/337700/#overview)  
[**2.1. public — всё**](https://habr.com/ru/post/337700/#o-01)  
[**2.2. Продвинутый assert**](https://habr.com/ru/post/337700/#o-02)  
[**2.3. Работа с исключениями**](https://habr.com/ru/post/337700/#o-03)  
[**2.4. Новый Test**](https://habr.com/ru/post/337700/#o-04)  
[**2.5. Новые базовые аннотации**](https://habr.com/ru/post/337700/#o-05)  
[**2.6. Вложенные классы**](https://habr.com/ru/post/337700/#o-06)  
[**2.7. Разделяемый инстанс класса для запуска тестов**](https://habr.com/ru/post/337700/#o-07)  
[**2.8. Автоматический повторный запуск теста**](https://habr.com/ru/post/337700/#o-08)  
[**2.9. Параметризированные тесты**](https://habr.com/ru/post/337700/#o-09)  
[**2.10. Аннотированные default методы в интерфейсах**](https://habr.com/ru/post/337700/#o-10)

**2.11. Остановка выполнения теста через необходимое время**  
[**3. Заключение**](https://habr.com/ru/post/337700/#conclusion)

1. Введение

Итак, официальный сайт начинает с того, что сообщает нам о новом строении JUnit:

JUnit 5 = JUnit Platform + JUnit Jupiter + JUnit Vintage (← [офф.сайт](http://junit.org/junit5/docs/current/user-guide/#overview-what-is-junit-5)).

***JUnit Platform*** — фундаментальная основа для запуска на JVM фреймворков для тестирования. Платформа предоставляет [TestEngine API](http://junit.org/junit5/docs/current/api/org/junit/platform/engine/TestEngine.html), для разработки фреймворков (для тестирования), которые могут быть запущены на платформе. Кроме этого, в платформе имеется [Console Launcher](http://junit.org/junit5/docs/current/user-guide/" \l "running-tests-console-launcher) для запуска платформы из коммандной строки а также для запуска любого JUnit 4 Runner'а на платформе. Уже, кстати, есть плагины для [Gradle](http://junit.org/junit5/docs/current/user-guide/" \l "running-tests-build-gradle) и [Maven](http://junit.org/junit5/docs/current/user-guide/" \l "running-tests-build-maven).  
  
***JUnit Jupiter*** — сердце JUnit 5. Этот проект предоставляет новые возможности для написания тестов и создания собственных расширений. В проекте реализован специальный TestEngine для запуска тестов на ранее описанной платформе.  
  
***JUnit Vintage*** — поддержка легаси. Определяется TestEngine для запуска тестов ориентированных на JUnit 3 и JUnit 4.

1. Начало работы

В интернете уже полно примеров для настройки [Gradle](https://github.com/junit-team/junit5-samples/tree/master/junit5-gradle-consumer) и [Maven](https://github.com/junit-team/junit5-samples/tree/master/junit5-maven-consumer) проектов. В блоге JetBrains есть отдельный пост, посвященный настройке [JUnit 5 в IDEA](https://blog.jetbrains.com/idea/2016/08/using-junit-5-in-intellij-idea/).

2. Обзор нововведений

А теперь наконец-то перейдем к примерам!  
  
**2.1. public — всё**  
JUnit больше не требует, чтобы методы были публичными.  
  
@Test

void test() {

assertEquals("It " + " works!" == "It works!");

}

**2.2. Продвинутый assert**  
Опциональное сообщение сделали последним аргументом.

assertEquals(2017, 2017, "The optional assertion message is now the last parameter.");

В пятой версии для конструирования сообщения можно использовать Supplier<String>.

assertTrue("habr" == "habr", () -> "Assertion messages can be lazily evaluated");

Добавили специальный метод для логической группировки тестов.  
  
// в группе все ассерты исполняются независимо,

// успех - когда прошли успешно все ассерты

assertAll("habr",

() -> assertThat("https://habrahabr.ru", startsWith("https")),

() -> assertThat("https://habrahabr.ru", endsWith(".ru"))

);

Появился метод для работы с Iterable.

assertIterableEquals(asList(1, 2, 3), asList(1, 2, 3));

Добавили интересный метод для сравнения набора строк. Поддерживаются регулярные выражения!  
  
Assertions.assertLinesMatch(

asList("можно сравнивать строки", "а можно по regex: \\d{2}\\.\\d{2}\\.\\d{4}"),

asList("можно сравнивать строки", "а можно по regex: 12.09.2017")

);

**2.3. Работа с исключениями**  
Работа с исключениями стала более линейной.  
  
Throwable exception = assertThrows(IllegalArgumentException.class, () -> {

throw new IllegalArgumentException("что-то пошло не так");

});

assertEquals("что-то пошло не так", exception.getMessage());

**2.4. Новый Test**  
JUnit 5 привнес новую аннотацию [Test](https://habrahabr.ru/users/test/), которая находится в пакете *org.junit.jupiter.api.Test*. В отличии от четвертой версии, новая аннотация служит исключительно маркером.

Посмотреть различия

// JUnit 4

@Retention(RetentionPolicy.RUNTIME)

@Target({ElementType.METHOD})

public @interface Test {

Class<? extends Throwable> expected() default Test.None.class;

long timeout() default 0L;

public static class None extends Throwable {

private static final long serialVersionUID = 1L;

private None() {

}

}

}

Новая аннотация выглядит так.  
  
/ JUnit 5

@Target({ ElementType.ANNOTATION\_TYPE, ElementType.METHOD })

@Retention(RetentionPolicy.RUNTIME)

@Documented

@API(Stable)

@Testable

public @interface Test {

}

**2.5. Новые базовые аннотации**  
В пятой версии добавили новые *базовые* аннотации.

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

import org.junit.jupiter.api.AfterAll;

import org.junit.jupiter.api.AfterEach;

import org.junit.jupiter.api.BeforeAll;

import org.junit.jupiter.api.BeforeEach;

import org.junit.jupiter.api.Disabled;

import org.junit.jupiter.api.Test;

class StandardTests {

// вместо @**BeforeClass**

@***BeforeAll***

static void initAll() {

}

// вместо @**Before**

@***BeforeEach***

void init() {

}

@Test

void succeedingTest() {

}

@Test

void failingTest() {

fail("a failing test");

}

// Вместо @**Ignore**

@***Test***

@***Disabled***("for demonstration purposes")

void skippedTest() {

// not executed

}

// Новая аннотация для улучшения читаемости при выводе результатов тестов.

@DisplayName("╯°□°）╯")

void testWithDisplayNameContainingSpecialCharacters() {}

// вместо @**After**

@***AfterEach***

void tearDown() {

}

// вместо @**AfterClass**

@***AfterAll***

static void tearDownAll() {

}

}

**2.6. Вложенные классы**  
Аннотация @Nested позволяет использовать внутренние классы при разработке тестов, что позволяет иногда более удобным способом группировать/дополнять тесты.

Пример из официальной документации.

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

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

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

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

import java.util.EmptyStackException;

import java.util.Stack;

import org.junit.jupiter.api.BeforeEach;

import org.junit.jupiter.api.DisplayName;

import org.junit.jupiter.api.Nested;

import org.junit.jupiter.api.Test;

@DisplayName("A stack")

class TestingAStackDemo {

Stack<Object> stack;

@Test

@DisplayName("is instantiated with new Stack()")

void isInstantiatedWithNew() {

new Stack<>();

}

@Nested

@DisplayName("when new")

class WhenNew {

@BeforeEach

void createNewStack() {

stack = new Stack<>();

}

@Test

@DisplayName("is empty")

void isEmpty() {

assertTrue(stack.isEmpty());

}

@Test

@DisplayName("throws EmptyStackException when popped")

void throwsExceptionWhenPopped() {

assertThrows(EmptyStackException.class, () -> stack.pop());

}

@Test

@DisplayName("throws EmptyStackException when peeked")

void throwsExceptionWhenPeeked() {

assertThrows(EmptyStackException.class, () -> stack.peek());

}

@Nested

@DisplayName("after pushing an element")

class AfterPushing {

String anElement = "an element";

@BeforeEach

void pushAnElement() {

stack.push(anElement);

}

@Test

@DisplayName("it is no longer empty")

void isNotEmpty() {

assertFalse(stack.isEmpty());

}

@Test

@DisplayName("returns the element when popped and is empty")

void returnElementWhenPopped() {

assertEquals(anElement, stack.pop());

assertTrue(stack.isEmpty());

}

@Test

@DisplayName("returns the element when peeked but remains not empty")

void returnElementWhenPeeked() {

assertEquals(anElement, stack.peek());

assertFalse(stack.isEmpty());

}

}

}

}

**2.7. Разделяемый инстанс класса для запуска тестов**  
Для гарантии независимости и изоляциии тестов JUnit во всех предыдущих версиях всегда создавал по инстансу на тест (т.е. на каждый запуск метода отдельный инстанс). В пятой версии такое поведение можно изменить используя новую аннотацию @TestInstance(Lifecycle.PER\_CLASS). В таком случае инстанс будет создан только один раз и будет переиспользован для запуска всех тестов, определенных внутри этого класса.  
  
**2.8. Автоматический повторный запуск теста**  
Еще одна приятная добавка! Аннотация @RepeatedTest сообщает JUnit, что данный тест нужно запустить несколько раз. При этом, каждый такой вызов будет независимым тестом, а значит для него будут работать аннотации @BeforeAll, @BeforeEach, @AfterEach и @AfterAll.  
  
@RepeatedTest(5)

void repeatedTest() {

System.out.println("Этот тест будет запущен пять раз. ");

}

Стоит отметить, что можно настроить дополнительный вывод информации о запусках теста. Например, показывать номер запуска. За это отвечают специальные константы определенные внутри этой же аннотации.  
  
**2.9. Параметризированные тесты**  
Параметризированные тесты позволяют запускать тест несколько раз с различными входными данными. На данный момент поддерживаются только данные примитивных типов: *int*, *long*, *double*, *String*. Но не стоит отчаиваться! JUnit 5 определяет несколько дополнительных аннотаций для указания источника данных для параметризированных тестов. Итак, начнём!  
  
@ParameterizedTest

@ValueSource(strings = { "Hello", "World" })

void testWithStringParameter(String argument) {

assertNotNull(argument);

}

Еще один вдохновляющий пример с @ValueSource.  
@ParameterizedTest

@ValueSource(strings = { "01.01.2017", "31.12.2017" })

void testWithConverter(@JavaTimeConversionPattern("dd.MM.yyyy") LocalDate date) {

assertEquals(2017, date.getYear());

}

Пример с разбором CSV.  
@ParameterizedTest

@CsvSource({ "foo, 1", "bar, 2", "'baz, qux', 3" })

// или даже так: @CsvFileSource(resources = "/two-column.csv")

void testWithCsvSource(String first, int second) {

assertNotNull(first);

assertNotEquals(0, second);

}

Пример с Enum.  
@ParameterizedTest

@EnumSource(value = TimeUnit.class, names = { "DAYS", "HOURS" })

void testWithEnumSourceInclude(TimeUnit timeUnit) {

assertTrue(EnumSet.of(TimeUnit.DAYS, TimeUnit.HOURS).contains(timeUnit));

}

Пример с источником данных.  
  
@ParameterizedTest

@ArgumentsSource(MyArgumentsProvider.class)

void testWithArgumentsSource(String argument) {

assertNotNull(argument);

}

static class MyArgumentsProvider implements ArgumentsProvider {

@Override

public Stream<? extends Arguments> provideArguments(ExtensionContext context) {

return Stream.of("foo", "bar").map(Arguments::of);

}

}

Еще больше крутых примеров можно найти на официальном сайте в разделе [3.13. Parameterized Tests](http://junit.org/junit5/docs/current/user-guide/#writing-tests-parameterized-tests).  
  
**2.10. Аннотированные default методы в интерфейсах**  
JUnit теперь умеет работать с default методами в интерфейсах! Вот один из официальных примеров применения этого нововведения. Предлагаю посмотреть интересный пример с Equals Contract.  
  
public interface Testable<T> {

T createValue();

}

public interface EqualsContract<T> extends Testable<T> {

T createNotEqualValue();

@Test

default void valueEqualsItself() {

T value = createValue();

assertEquals(value, value);

}

@Test

default void valueDoesNotEqualNull() {

T value = createValue();

assertFalse(value.equals(null));

}

@Test

default void valueDoesNotEqualDifferentValue() {

T value = createValue();

T differentValue = createNotEqualValue();

assertNotEquals(value, differentValue);

assertNotEquals(differentValue, value);

}

}

**2.11. Если нужно остановить тест через необходимое время, то**

@Test

void getAllUsers() {

*//нужно писать assertTimeoutPreemptively() и тогда выполнение метода*

*//принудительно остановится через необходимое время с ошибкой и напишет наш текст*

assertTimeoutPreemptively(Duration.ofMillis(3), () -> {

*//если написать через assertTimeout(), то метод будет выполняться все*

*// необходимое ему время, а потом напишет наш текст*

assertTimeout(ofSeconds(1), () -> {

while (true); //или любой текст метода

},() -> "Тест выполняется больше 3 ms");

}

Заключение

Очень здорово, что популярный фреймворк для тестирования решается на такие серьезные эксперименты с API и старается идти в ногу со временем!  
  
Напоследок оставлю парочку ссылок: [официальный сайт JUnit 5](http://junit.org/junit5/) и [дружелюбное руководство](http://junit.org/junit5/docs/current/user-guide/).  
  
Еще много чего интересного осталось за рамками этой статьи. Например, отдельного обзора заслуживает механизм расширений, предоставляемый JUnit 5.  
Спасибо за внимание!