

# AssertJ User Guide

Version 1.0.0-SNAPSHOT

# Table of Contents

1. Overview .....	1
1.1. What is JUnit 5? .....	1
1.2. Supported Java Versions .....	1
1.3. Getting Help .....	1
2. Installation .....	1
2.1. Dependency Metadata .....	1
2.2. JUnit Jupiter Sample Projects .....	2
3. Writing Tests .....	2
3.1. Annotations .....	2
3.2. Test Classes and Methods .....	3
3.3. Display Names .....	5
3.4. Assertions .....	5
3.5. Assumptions .....	8
3.6. Disabling Tests .....	9
4. Contributors .....	10
5. Release Notes .....	10

# 1. Overview

The goal of this document is to provide comprehensive reference documentation for programmers writing tests, extension authors, and engine authors as well as build tool and IDE vendors.



## Translations

This document is also available in [Simplified Chinese](#) and [Japanese](#).

## 1.1. What is JUnit 5?

Unlike previous versions of JUnit, JUnit 5 is composed of several different modules from three different sub-projects.

JUnit Vintage provides a **TestEngine** for running JUnit 3 and JUnit 4 based tests on the platform.

## 1.2. Supported Java Versions

JUnit 5 requires Java 8 (or higher) at runtime. However, you can still test code that has been compiled with previous versions of the JDK.

## 1.3. Getting Help

Ask JUnit 5 related questions on [Stack Overflow](#) or chat with us on {Gitter}.

# 2. Installation

Artifacts for final releases and milestones are deployed to Maven Central.

Snapshot artifacts are deployed to Sonatype's {snapshot-repo}[snapshots repository] under {snapshot-repo}/org/junit/[org/junit].

## 2.1. Dependency Metadata

### 2.1.1. Dependencies

In addition, most of the above artifacts have a direct or transitive dependency to the following *OpenTest4J* JAR.

- **Group ID:** `org.opentest4j`
- **Artifact ID:** `opentest4j`
- **Version:** `{ota4j-version}`

## 2.2. JUnit Jupiter Sample Projects

The `{junit5-samples-repo}``{junit5-samples}` repository hosts a collection of sample projects based on JUnit Jupiter and JUnit Vintage. You'll find the respective `build.gradle` and `pom.xml` in the projects below.

- For Gradle, check out the `{junit5-jupiter-starter-gradle}` project.
- For Maven, check out the `{junit5-jupiter-starter-maven}` project.

## 3. Writing Tests

*A first test case*

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

import org.junit.jupiter.api.Test;

class FirstJUnit5Tests {

    @Test
    void myFirstTest() {
        assertEquals(2, 1 + 1);
    }

}
```

### 3.1. Annotations

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

All core annotations are located in the `{api-package}` package in the `junit-jupiter-api` module.

Annotation	Description
<code>@Test</code>	Denotes that a method is a test method. Unlike JUnit 4's <code>@Test</code> annotation, this annotation does not declare any attributes, since test extensions in JUnit Jupiter operate based on their own dedicated annotations. Such methods are <i>inherited</i> unless they are <i>overridden</i> .
<code>@Tag</code>	Used to declare <i>tags</i> for filtering tests, either at the class or method level; analogous to test groups in TestNG or Categories in JUnit 4. Such annotations are <i>inherited</i> at the class level but not at the method level.
<code>@Disabled</code>	Used to <i>disable</i> a test class or test method; analogous to JUnit 4's <code>@Ignore</code> . Such annotations are not <i>inherited</i> .

#### 3.1.1. Meta-Annotations and Composed Annotations

JUnit Jupiter annotations can be used as *meta-annotations*. That means that you can define your

own *composed annotation* that will automatically *inherit* the semantics of its meta-annotations.

```
import java.lang.annotation.ElementType;
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;
import java.lang.annotation.Target;

import org.junit.jupiter.api.Tag;

@Target({ ElementType.TYPE, ElementType.METHOD })
@Retention(RetentionPolicy.RUNTIME)
@Tag("fast")
public @interface Fast {
}
```

## 3.2. Test Classes and Methods

A *test method* is any instance method that is directly or meta-annotated with `@Test`, `@RepeatedTest`, `@ParameterizedTest`, `@TestFactory`, or `@TestTemplate`. A *test class* is any top level or static member class that contains at least one test method.

```
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 {

    @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
    }

    @AfterEach
    void tearDown() {
    }

    @AfterAll
    static void tearDownAll() {
    }

}
```



Neither test classes nor test methods need to be **public**.

## 3.3. Display Names

Test classes and test methods can declare custom display names — with spaces, special characters, and even emojis — that will be displayed by test runners and test reporting.

```
import org.junit.jupiter.api.DisplayName;
import org.junit.jupiter.api.Test;

@DisplayName("A special test case")
class DisplayNameDemo {

    @Test
    @DisplayName("Custom test name containing spaces")
    void testWithDisplayNameContainingSpaces() {
    }

    @Test
    @DisplayName(" °□° ")
    void testWithDisplayNameContainingSpecialCharacters() {
    }

    @Test
    @DisplayName(" ")
    void testWithDisplayNameContainingEmoji() {
    }
}
```

## 3.4. Assertions

JUnit Jupiter comes with many of the assertion methods that JUnit 4 has and adds a few that lend themselves well to being used with Java 8 lambdas. All JUnit Jupiter assertions are **static** methods in the `{Assertions}` class.

```
import static java.time.Duration.ofMillis;
import static java.time.Duration.ofMinutes;
import static org.junit.jupiter.api.Assertions.assertAll;
import static org.junit.jupiter.api.Assertions.assertEquals;
import static org.junit.jupiter.api.Assertions.assertNotNull;
import static org.junit.jupiter.api.Assertions.assertThrows;
import static org.junit.jupiter.api.Assertions.assertTimeout;
import static org.junit.jupiter.api.Assertions.assertTimeoutPreemptively;
import static org.junit.jupiter.api.Assertions.assertTrue;

import org.junit.jupiter.api.Test;

class AssertionsDemo {
```

```

@Test
void standardAssertions() {
    assertEquals(2, 2);
    assertEquals(4, 4, "The optional assertion message is now the last parameter.");
    assertTrue('a' < 'b', () -> "Assertion messages can be lazily evaluated -- "
        + "to avoid constructing complex messages unnecessarily.");
}

@Test
void groupedAssertions() {
    // In a grouped assertion all assertions are executed, and any
    // failures will be reported together.
    assertAll("person",
        () -> assertEquals("John", person.getFirstName()),
        () -> assertEquals("Doe", person.getLastName())
    );
}

@Test
void dependentAssertions() {
    // Within a code block, if an assertion fails the
    // subsequent code in the same block will be skipped.
    assertAll("properties",
        () -> {
            String firstName = person.getFirstName();
            assertNotNull(firstName);

            // Executed only if the previous assertion is valid.
            assertAll("first name",
                () -> assertTrue(firstName.startsWith("J")),
                () -> assertTrue(firstName.endsWith("n"))
            );
        },
        () -> {
            // Grouped assertion, so processed independently
            // of results of first name assertions.
            String lastName = person.getLastName();
            assertNotNull(lastName);

            // Executed only if the previous assertion is valid.
            assertAll("last name",
                () -> assertTrue(lastName.startsWith("D")),
                () -> assertTrue(lastName.endsWith("e"))
            );
        }
    );
}

@Test
void exceptionTesting() {
    Throwable exception = assertThrows(IllegalArgumentException.class, () -> {

```



```

        throw new IllegalArgumentException("a message");
    });
    assertEquals("a message", exception.getMessage());
}

@Test
void timeoutNotExceeded() {
    // The following assertion succeeds.
    assertTimeout(ofMinutes(2), () -> {
        // Perform task that takes less than 2 minutes.
    });
}

@Test
void timeoutNotExceededWithResult() {
    // The following assertion succeeds, and returns the supplied object.
    String actualResult = assertTimeout(ofMinutes(2), () -> {
        return "a result";
    });
    assertEquals("a result", actualResult);
}

@Test
void timeoutNotExceededWithMethod() {
    // The following assertion invokes a method reference and returns an object.
    String actualGreeting = assertTimeout(ofMinutes(2), AssertionsDemo::greeting);
    assertEquals("Hello, World!", actualGreeting);
}

@Test
void timeoutExceeded() {
    // The following assertion fails with an error message similar to:
    // execution exceeded timeout of 10 ms by 91 ms
    assertTimeout(ofMillis(10), () -> {
        // Simulate task that takes more than 10 ms.
        Thread.sleep(100);
    });
}

@Test
void timeoutExceededWithPreemptiveTermination() {
    // The following assertion fails with an error message similar to:
    // execution timed out after 10 ms
    assertTimeoutPreemptively(ofMillis(10), () -> {
        // Simulate task that takes more than 10 ms.
        Thread.sleep(100);
    });
}

private static String greeting() {
    return "Hello, World!";
}

```

```
}  
  
}
```

JUnit Jupiter also comes with a few assertion methods that lend themselves well to being used in [Kotlin](#). All JUnit Jupiter Kotlin assertions are top-level functions in the `org.junit.jupiter.api` package.

### 3.4.1. Third-party Assertion Libraries

Even though the assertion facilities provided by JUnit Jupiter are sufficient for many testing scenarios, there are times when more power and additional functionality such as *matchers* are desired or required. In such cases, the JUnit team recommends the use of third-party assertion libraries such as [AssertJ](#), {Hamcrest}, {Truth}, etc. Developers are therefore free to use the assertion library of their choice.

For example, the combination of *matchers* and a fluent API can be used to make assertions more descriptive and readable. However, JUnit Jupiter's `{Assertions}` class does not provide an `assertThat()` method like the one found in JUnit 4's `org.junit.Assert` class which accepts a Hamcrest *Matcher*. Instead, developers are encouraged to use the built-in support for matchers provided by third-party assertion libraries.

The following example demonstrates how to use the `assertThat()` support from Hamcrest in a JUnit Jupiter test. As long as the Hamcrest library has been added to the classpath, you can statically import methods such as `assertThat()`, `is()`, and `equalTo()` and then use them in tests like in the `assertWithHamcrestMatcher()` method below.

```
import static org.hamcrest.CoreMatchers.equalTo;  
import static org.hamcrest.CoreMatchers.is;  
import static org.hamcrest.MatcherAssert.assertThat;  
  
import org.junit.jupiter.api.Test;  
  
class HamcrestAssertionDemo {  
  
    @Test  
    void assertWithHamcrestMatcher() {  
        assertThat(2 + 1, is(equalTo(3)));  
    }  
  
}
```

Naturally, legacy tests based on the JUnit 4 programming model can continue using `org.junit.Assert#assertThat`.

## 3.5. Assumptions

JUnit Jupiter comes with a subset of the assumption methods that JUnit 4 provides and adds a few

that lend themselves well to being used with Java 8 lambdas. All JUnit Jupiter assumptions are static methods in the `{Assumptions}` class.

```
import static org.junit.jupiter.api.Assertions.assertEquals;
import static org.junit.jupiter.api.Assumptions.assumeTrue;
import static org.junit.jupiter.api.Assumptions.assumingThat;

import org.junit.jupiter.api.Test;

class AssumptionsDemo {

    @Test
    void testOnlyOnCiServer() {
        assumeTrue("CI".equals(System.getenv("ENV")));
        // remainder of test
    }

    @Test
    void testOnlyOnDeveloperWorkstation() {
        assumeTrue("DEV".equals(System.getenv("ENV")),
            () -> "Aborting test: not on developer workstation");
        // remainder of test
    }

    @Test
    void testInAllEnvironments() {
        assumingThat("CI".equals(System.getenv("ENV")),
            () -> {
                // perform these assertions only on the CI server
                assertEquals(2, 2);
            });

        // perform these assertions in all environments
        assertEquals("a string", "a string");
    }
}
```

## 3.6. Disabling Tests

Entire test classes or individual test methods may be *disabled* via the `{Disabled}` annotation, via one of the annotations discussed in

Here's a `@Disabled` test class.

```
import org.junit.jupiter.api.Disabled;
import org.junit.jupiter.api.Test;

@Disabled
class DisabledClassDemo {
    @Test
    void testWillBeSkipped() {
    }
}
```

And here's a test class that contains a `@Disabled` test method.

```
import org.junit.jupiter.api.Disabled;
import org.junit.jupiter.api.Test;

class DisabledTestsDemo {

    @Disabled
    @Test
    void testWillBeSkipped() {
    }

    @Test
    void testWillBeExecuted() {
    }
}
```

## 4. Contributors

Browse the [current list of contributors](#) directly on GitHub.

## 5. Release Notes

The release notes are available [here](#).