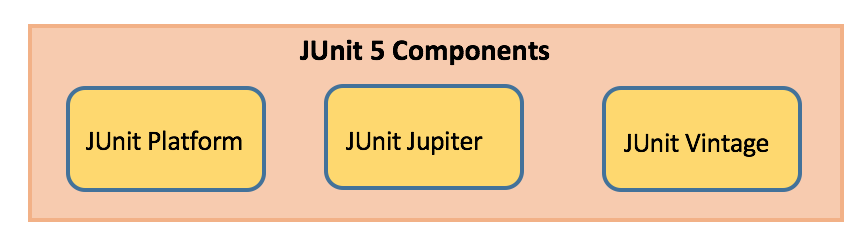
# JUnit5 Tutorial

## JUnit5 Tutorial

In this Junit tutorial, we will introduce basics of JUnit5 and its new features using examples. In Java world, JUnit is one of the popular framework used to implement unit tests against java code. JUnit primarily helps developers to test their code on the [JVM](https://www.digitalocean.com/community/tutorials/java-jvm-memory-model-memory-management-in-java) by themselves.

## JUnit5 Architecture



### JUnit Platform

* Launches testing frameworks on the JVM
* Has TestEngine API used to build a testing framework that runs on the JUnit platform

### JUnit Jupiter

* Blend of new programming model for writing tests and extension model for extensions
* Addition of new [annotations](https://www.digitalocean.com/community/tutorials/java-annotations) like @BeforeEach, @AfterEach, @AfterAll, @BeforeAll etc.

### JUnit Vintage

* Provides support to execute previous JUnit version 3 and 4 tests on this new platform

## JUnit Maven Dependencies

To implement JUnit5 based test cases in a project, add the following dependency to the pom.xml file of the project:

* JUnit 5 Library

<dependency>

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

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

<version>5.1.1</version>

<scope>test</scope>

</dependency>

<dependency>

<groupId>org.junit.platform</groupId>

<artifactId>junit-platform-runner</artifactId>

<version> 1.1.1</version>

<scope>test</scope>

</dependency>

* JUnit5 maven surefire provider to execute the unit tests where IDE does not have JUnit5 support (if IDE has support then this point is not required)

<plugin>

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

<version>2.19.1</version>

<dependencies>

<dependency>

<groupId>org.junit.platform</groupId>

<artifactId>junit-platform-surefire-provider</artifactId>

<version>1.0.2</version>

</dependency>

</dependencies>

</plugin>

## JUnit5 New Features

It requires Java 8 or higher at runtime. But one can still test code which is compiled using previous Java versions. There are various new features got introduced in it.

### JUnit Annotations

Listed below are some commonly used annotations provided in it:

| **Annotation** | **Description** |
| --- | --- |
| [@Test](https://www.digitalocean.com/community/users/test) | Denotes a test method |
| [@DisplayName](https://www.digitalocean.com/community/users/displayname) | Declares a custom display name for the test class or test method |
| [@BeforeEach](https://www.digitalocean.com/community/users/beforeeach) | Denotes that the annotated method should be executed before each test method |
| [@AfterEach](https://www.digitalocean.com/community/users/aftereach) | Denotes that the annotated method should be executed after each test method |
| [@BeforeAll](https://www.digitalocean.com/community/users/beforeall) | Denotes that the annotated method should be executed before all test methods |
| [@AfterAll](https://www.digitalocean.com/community/users/afterall) | Denotes that the annotated method should be executed after all test methods |
| [@Disable](https://www.digitalocean.com/community/users/disable) | Used to disable a test class or test method |
| [@Nested](https://www.digitalocean.com/community/users/nested) | Denotes that the annotated class is a nested, non-static test class |
| [@Tag](https://www.digitalocean.com/community/users/tag) | Declare tags for filtering tests |
| [@ExtendWith](https://www.digitalocean.com/community/users/extendwith) | Register custom extensions |

package com.journaldev;

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.DisplayName;

import org.junit.jupiter.api.Test;

public class JUnit5Sample1Test {

@BeforeAll

static void before() {

System.out.println("\*\*--- Executed once before all test methods in this class ---\*\*");

}

@BeforeEach

void beforeEach() {

System.out.println("\*\*--- Executed before each test method in this class ---\*\*");

}

@Test

void testMethod1() {

System.out.println("\*\*--- Test method1 executed ---\*\*");

}

@DisplayName("Test method2 with condition")

@Test

void testMethod2() {

System.out.println("\*\*--- Test method2 executed ---\*\*");

}

@Test

@Disabled("implementation pending")

void testMethod3() {

System.out.println("\*\*--- Test method3 executed ---\*\*");

}

@AfterEach

void afterEach() {

System.out.println("\*\*--- Executed after each test method in this class ---\*\*");

}

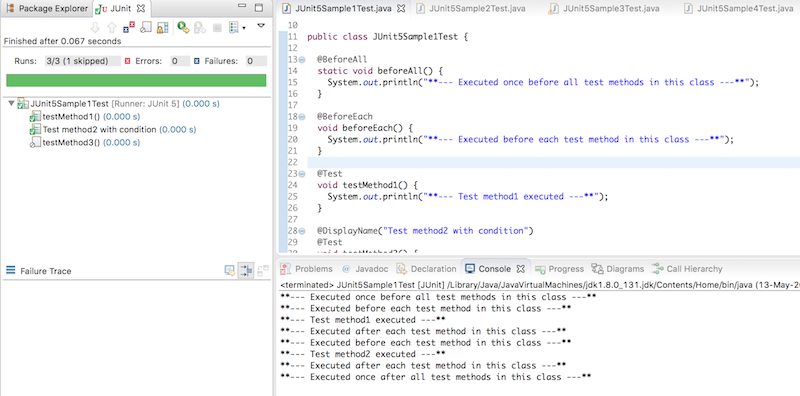
@AfterAll

static void afterAll() {

System.out.println("\*\*--- Executed once after all test methods in this class ---\*\*");

}

}

We can run above JUnit test class in **Eclipse -> Run As -> JUnit Test**.

### JUnit Assertions

Every test method must be evaluated against condition to true using assertions so that the test can continue to execute. JUnit Jupiter assertions are kept in the org.junit.jupiter.api.Assertions class. All of the methods are static.

| **Assertion** | **Description** |
| --- | --- |
| assertEquals(expected, actual) | Fails when expected does not equal actual |
| assertFalse(expression) | Fails when expression is not false |
| assertNull(actual) | Fails when actual is not null |
| assertNotNull(actual) | Fails when actual is null |
| assertAll() | Group many assertions and every assertion is executed even if one or  more of them fails |
| assertTrue(expression) | Fails if expression is not true |
| assertThrows() | Class to be tested is expected to throw an exception |

@Test

void testAssertEqual() {

assertEquals("ABC", "ABC");

assertEquals(20, 20, "optional assertion message");

assertEquals(2 + 2, 4);

}

@Test

void testAssertFalse() {

assertFalse("FirstName".length() == 10);

assertFalse(10 > 20, "assertion message");

}

@Test

void testAssertNull() {

String str1 = null;

String str2 = "abc";

assertNull(str1);

assertNotNull(str2);

}

@Test

void testAssertAll() {

String str1 = "abc";

String str2 = "pqr";

String str3 = "xyz";

assertAll("numbers",

() -> assertEquals(str1,"abc"),

() -> assertEquals(str2,"pqr"),

() -> assertEquals(str3,"xyz")

);

//uncomment below code and understand each assert execution

/\*assertAll("numbers",

() -> assertEquals(str1,"abc"),

() -> assertEquals(str2,"pqr1"),

() -> assertEquals(str3,"xyz1")

);\*/

}

@Test

void testAssertTrue() {

assertTrue("FirstName".startsWith("F"));

assertTrue(10 {

throw new IllegalArgumentException("Illegal Argument Exception occured");

});

assertEquals("Illegal Argument Exception occured", exception.getMessage());

}

### JUnit5 Imports

Its test classes need org.junit.jupiter.api.Test import statement and not org.junit.Test. Also, the test methods need not be a public and local package.

import org.junit.jupiter.api.Test;

### JUnit5 Assumptions

Assumptions are static methods in the org.junit.jupiter.api.Assumptions class. They will execute a test only when the specified condition met otherwise test will be aborted. The aborted test will not cause build failure. When an assumption fails, org.opentest4j.TestAbortedException is thrown and the test is skipped.

| **Assumptions** | **Description** |
| --- | --- |
| assumeTrue | Execute the body of lamda when the positive condition hold else test will be skipped |
| assumeFalse | Execute the body of lamda when the negative condition hold else test will be skipped |
| assumingThat | Portion of the test method will execute if an assumption holds true and everything after the lambda will execute irrespective of the assumption in assumingThat() holds |

@Test

void testAssumeTrue() {

boolean b = 'A' == 'A';

assumeTrue(b);

assertEquals("Hello", "Hello");

}

@Test

@DisplayName("test executes only on Saturday")

public void testAssumeTrueSaturday() {

LocalDateTime dt = LocalDateTime.now();

assumeTrue(dt.getDayOfWeek().getValue() == 6);

System.out.println("further code will execute only if above assumption holds true");

}

@Test

void testAssumeFalse() {

boolean b = 'A' != 'A';

assumeFalse(b);

assertEquals("Hello", "Hello");

}

@Test

void testAssumeFalseEnvProp() {

System.setProperty("env", "prod");

assumeFalse("dev".equals(System.getProperty("env")));

System.out.println("further code will execute only if above assumption hold");

}

@Test

void testAssumingThat() {

System.setProperty("env", "test");

assumingThat("test".equals(System.getProperty("env")),

() -> {

assertEquals(10, 10);

System.out.println("perform below assertions only on the test env");

});

assertEquals(20, 20);

System.out.println("perform below assertions on all env");

}