1. @DisplayName

1.1 Default name of test classes and methods.

DisplayNameTest.java

package com.mkyong.display;

import org.junit.jupiter.api.Test;

public class DisplayNameTest {

@Test

void test\_spaces\_ok() {

}

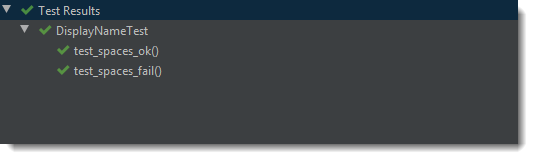
@Test

void test\_spaces\_fail() {

}

}

Output



+-- JUnit Jupiter [OK]

| '-- DisplayNameTest [OK]

| +-- test\_spaces\_ok() [OK]

| '-- test\_spaces\_fail() [OK]

1.2 @DisplayName

DisplayNameCustomTest.java

package com.mkyong.display;

import org.junit.jupiter.api.DisplayName;

import org.junit.jupiter.api.Test;

@DisplayName("I'm a Test Class")

public class DisplayNameCustomTest {

@Test

@DisplayName("Test with spaces, expected ok")

void test\_spaces\_ok() {

}

@DisplayName("Test with spaces, expected failed")

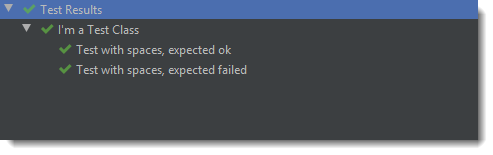
@Test

void test\_spaces\_fail() {

}

}

Output



+-- JUnit Jupiter [OK]

| '-- I'm a Test Class [OK]

| +-- Test with spaces, expected ok [OK]

| '-- Test with spaces, expected failed [OK]

2. Display Name Generators

2.1 We can also create a custom display name generator and configured via the @DisplayNameGeneration.

2.2 This example uses the JUnit ReplaceUnderscores generator to replace the underscores with spaces.

DisplayNameGenerator1Test.java

package com.mkyong.display;

import org.junit.jupiter.api.DisplayNameGeneration;

import org.junit.jupiter.api.DisplayNameGenerator;

import org.junit.jupiter.api.Test;

import java.lang.reflect.Method;

@DisplayNameGeneration(DisplayNameGenerator.ReplaceUnderscores.class)

public class DisplayNameGenerator1Test {

@Test

void test\_spaces\_ok() {

}

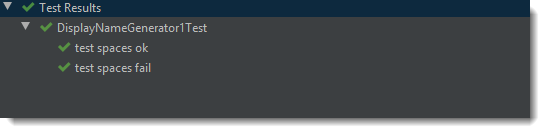
@Test

void test\_spaces\_fail() {

}

}

Output



2.3 We can extend the JUnit DisplayNameGenerator to create our custom display name generator.

DisplayNameGenerator2Test.java

package com.mkyong.display;

import org.junit.jupiter.api.DisplayNameGeneration;

import org.junit.jupiter.api.DisplayNameGenerator;

import org.junit.jupiter.api.Test;

import java.lang.reflect.Method;

import java.util.Arrays;

import java.util.stream.Collectors;

@DisplayNameGeneration(DisplayNameGenerator2Test.CustomDisplayNameGenerator.class)

public class DisplayNameGenerator2Test {

@Test

void test\_spaces\_ok() {

}

@Test

void test\_spaces\_fail() {

}

static class CustomDisplayNameGenerator extends DisplayNameGenerator.Standard {

@Override

public String generateDisplayNameForClass(Class<?> testClass) {

return "New Name for test class";

}

@Override

public String generateDisplayNameForNestedClass(Class<?> nestedClass) {

return super.generateDisplayNameForNestedClass(nestedClass);

}

@Override

public String generateDisplayNameForMethod(Class<?> testClass, Method testMethod) {

String name = testMethod.getName();

return Arrays.stream(name.split("\_")).collect(Collectors.joining(" | "));

}

}

}

Output



3. Parameterized Tests

3.1 For parameterized tests, we can declare the custom display name via the name attribute of the @ParameterizedTest, see the following example:

DisplayNameParamTest.java

package com.mkyong.display;

import org.junit.jupiter.params.ParameterizedTest;

import org.junit.jupiter.params.provider.Arguments;

import org.junit.jupiter.params.provider.EnumSource;

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

import java.util.concurrent.TimeUnit;

import java.util.stream.Stream;

import static org.junit.jupiter.params.provider.Arguments.arguments;

public class DisplayNameParamTest {

@ParameterizedTest(name = "#{index} - Test with TimeUnit: {0}")

@EnumSource(value = TimeUnit.class, names = {"MINUTES", "SECONDS"})

void test\_timeunit\_ok(TimeUnit time) {

}

@ParameterizedTest(name = "#{index} - Test with {0} and {1}")

@MethodSource("argumentProvider")

void test\_method\_multi(String str, int length) {

}

static Stream<Arguments> argumentProvider() {

return Stream.of(

arguments("abc", 3),

arguments("lemon", 2)

);

}

}

Output

