

ParameterizedTest with @MethodSource

This lesson demonstrates use of @MethodSource to pass different arguments to @ParameterizedTest.

We'll cover the following



- @MethodSource

@MethodSource

`@MethodSource` allows us to specify a factory method for different test arguments. This factory method must be `static` and can be present in the same class or any other class too. The factory method should return Stream, Iterator, Iterable or array of elements to `@ParameterizedTest` method.

Let's look at a demo.

Step 1 - Let's assume that we have to write a parameterized test that takes values of the `static` factory method as `@MethodSource`.

Step 2 - We create a test class by name, `MethodSourceTest.java`.

Step 3 - It contains a test method by name, `testMethodSource`. In order to provide different parameters/values to the same test method, this method is marked as `@ParameterizedTest` instead of `@Test`.

Step 4 - In order to provide different and multiple values through factory methods. We mark this test method with `@MethodSource` annotation. This annotation takes in name of factory method which will provide streams/lists of data to `@ParameterizedTest`.

Let's see the test class below.

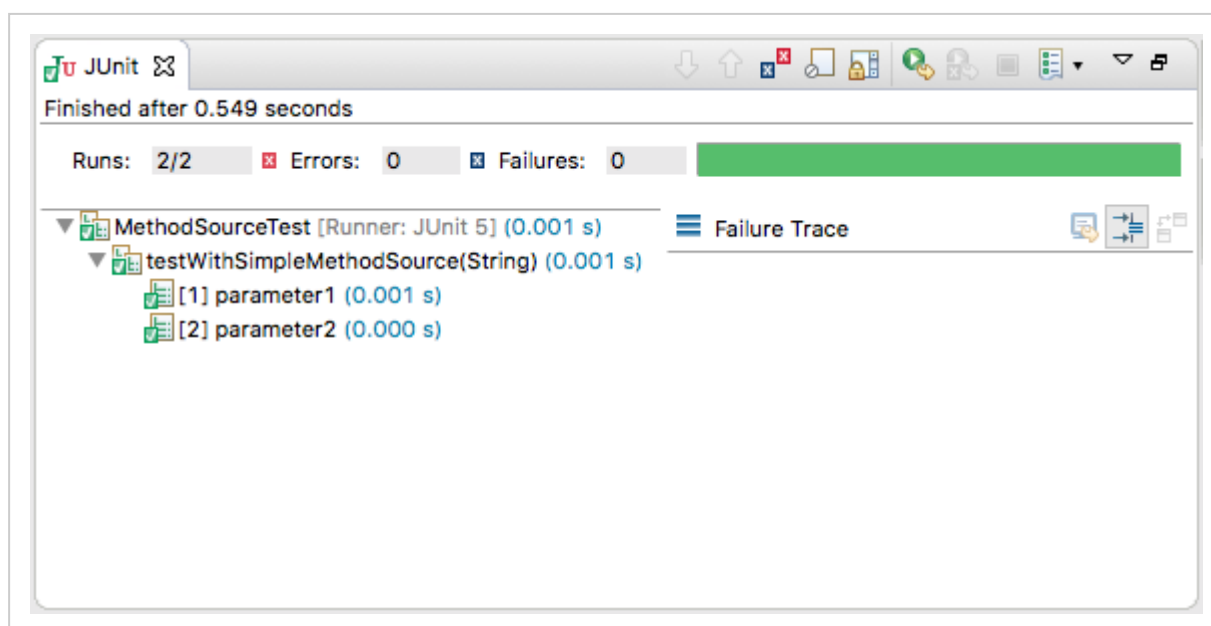
```
1 package io.educative.junit5;  
2  
3 import static org.junit.jupiter.api.Assertions.*;  
4
```



```

5  import java.util.stream.Stream;
6
7  import org.junit.jupiter.params.ParameterizedTest;
8  import org.junit.jupiter.params.provider.MethodSource;
9
10 class MethodSourceTest {
11
12     @ParameterizedTest
13     @MethodSource("parameterProvider")
14     void testWithSimpleMethodSource(String argument) {
15         assertNotNull(argument);
16     }
17
18     // method name is the source to test
19     static Stream<String> parameterProvider() {
20         return Stream.of("parameter1", "parameter2");
21     }
22
23 }

```



Output of @ParameterizedTest demo

Above image demonstrates the working of `@ParameterizedTest`. As we have provided two different method source values by providing a `static` method as `parameterProvider()`, whose return type is Streams of String, therefore, the test case ran 2 times. We provide a static method name to `@MethodSource` annotation, which internally calls method `parameterProvider()` to get different parameters values. Also, all string values provided by

`parameterProvider()` method are not null therefore `assertNotNull` passes for all values passed.

In the next lesson we will be studying parameterized tests with `@CsvSource`.