ParameterizedTest with @CsvSource

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

We'll cover the following

@CsvSource

`@CsvSource` allows you to provide parameter lists as comma-separated custom-delimiter separated values. `@CsvSource` annotation uses single quote along with comma-separated delimiter to distinguish a csv value from others.

For e.g -

- {“one, two”} will result to 2 arguments as - “one”, “two”.
- {“one, ‘two, three’”} will result to 2 arguments as - “one”, “two, three”.
- {“one, “”} will result to 2 arguments as - “one”, “”.
- {"one, "} will result to 2 arguments as - “one”, null.

Let’s look at a demo.

Step 1 - Let’s assume that we have to write a parameterized test that takes values from `@CsvSource`.

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

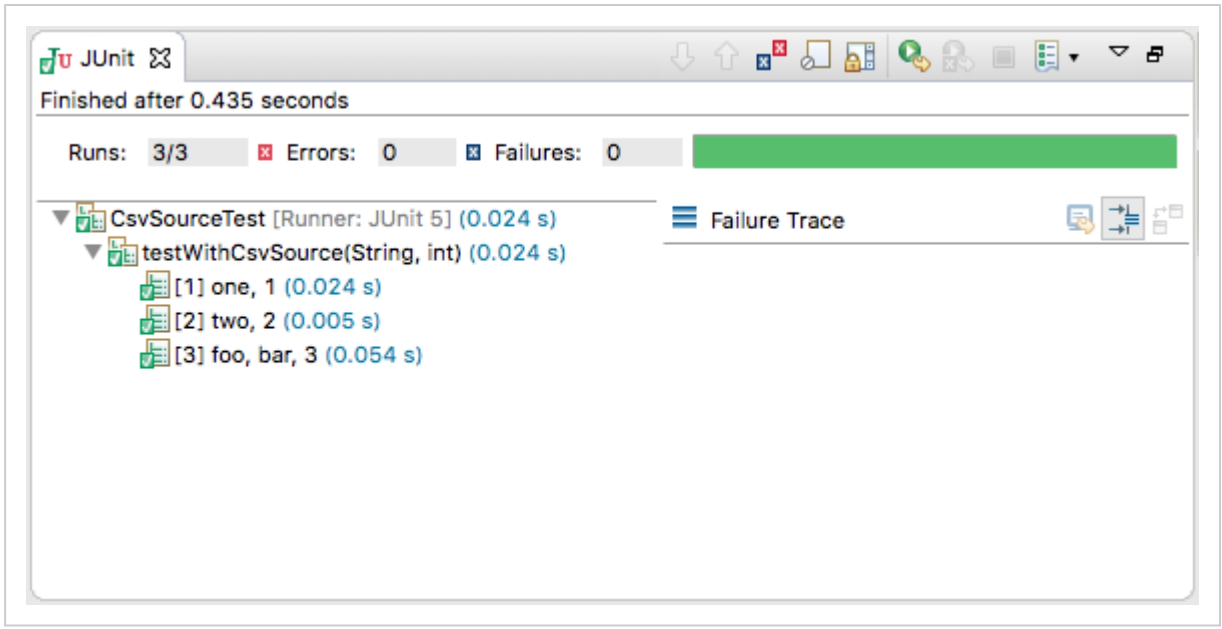
Step 3 - It contains a test method by name, `testCsvSource`. In order to provide different parameters/values to the same test method, this method is marked as `@ParameterizedTest` instead of `@Test`.

Step 4 - In order to provide different and multiple values through csv source. We mark this test method with `@CsvSource` annotation. This annotation takes comma-separated values 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.jupite
4
5  import org.junit.jupiter.param
6  import org.junit.jupiter.param
7
8  class CsvSourceTest {
9
10     @ParameterizedTest
11     @CsvSource({ "one, 1", "tw
12     void testWithCsvSource(Str
13         assertNotNull(first);
14         assertNotEquals(0, sec
15     }
16
17 }
```



Output of @ParameterizedTest demo

Above image demonstrates the working of `@ParameterizedTest`. As we have provided 3 different csv source values which are comma-separated, so the first argument to test method is a String and the second argument is an integer type, therefore the test case ran 3 times. Also, all string and integer values provided by csv source are not null and integer value is not 0.

values provided by csv source are not null and integer value is not 0, therefore `assertNotNull` and `assertNotEquals` passes for all values passed.

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