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`.

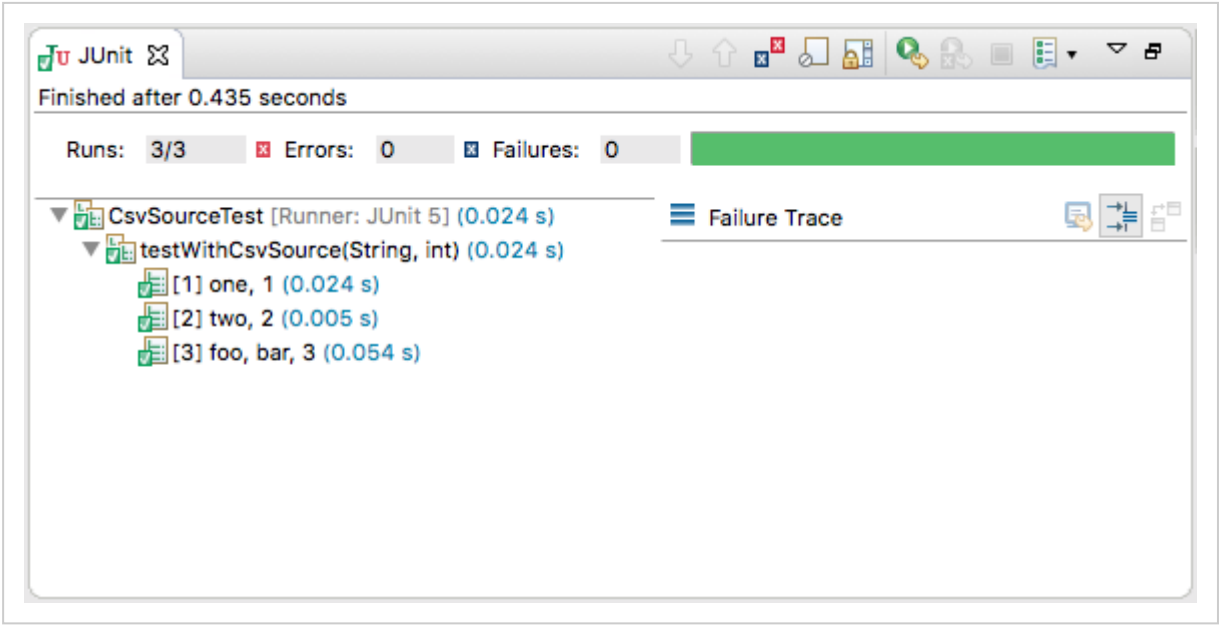
Step 3 - It contains a test method by name, `testCsvSource`. In order to provide different parameters/values to the same test method, this method is marked as `@ParameterizedTest` instead of `@Test`.

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.

```
package io.educative.junit5;

import static org.junit.jupiter.api.Assertions.*;

import org.junit.jupiter.params.ParameterizedTest;
import org.junit.jupiter.params.provider.CsvSource;

class CsvSourceTest {

        @ParameterizedTest
        @CsvSource({ "one, 1", "two, 2", "'foo, bar', 3" })
        void testWithCsvSource(String first, int second) {
            assertNotNull(first);
            assertNotEquals(0, second);
        }

}
```



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, therefore `assertNotNull` and `assertNotEquals` passes for all values passed.

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