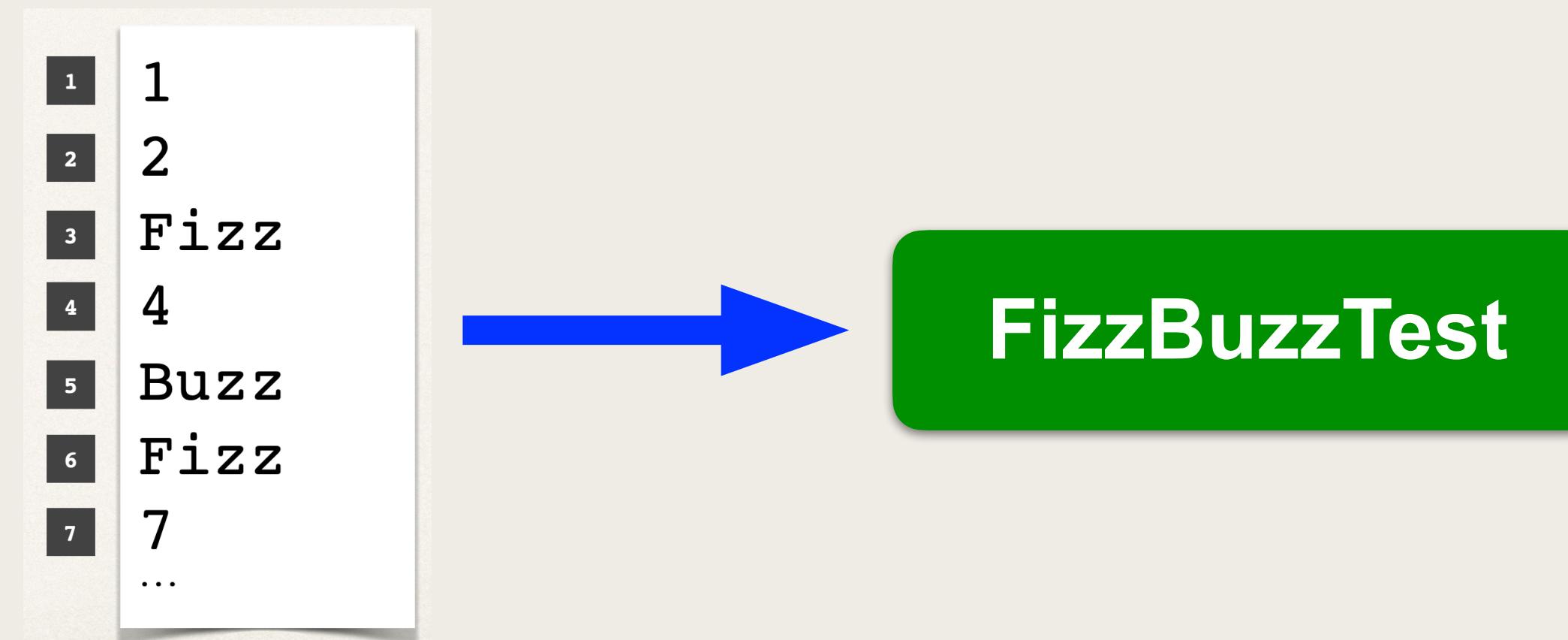


# Parameterized Tests



# Fizz Buzz Input Values

- At the moment we have created tests for specific FizzBuzz input values
- We'd like to pass in a collection of values and expected results
- Run the same test in a loop

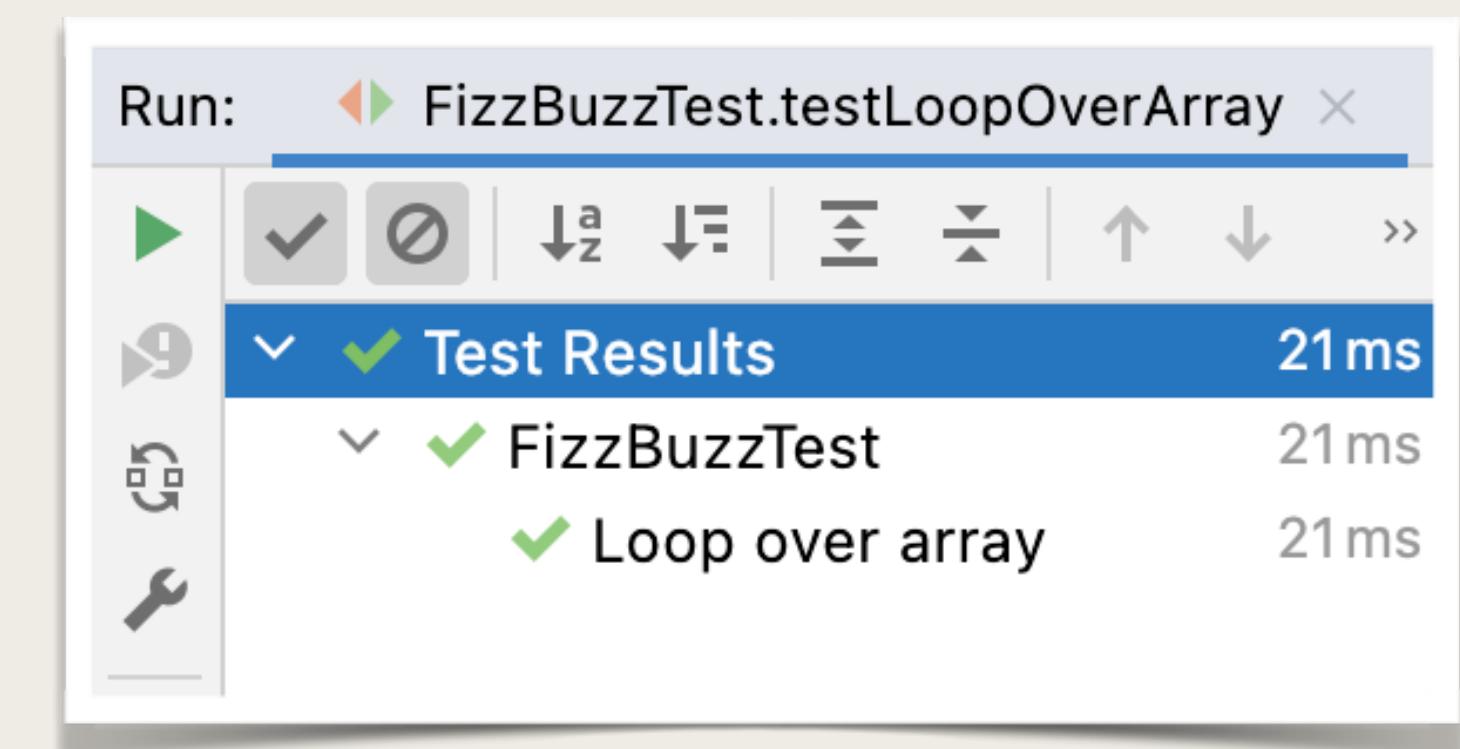


# One Possible Solution

```
@DisplayName("Loop over array")
@Test
@Order(5)
void testLoopOverArray() {
    String[][] data = { {"1", "1"}, {"2", "2"}, {"3", "Fizz"}, {"4", "4"}, {"5", "Buzz"}, {"5", "Fizz"}, {"7", "7"} };
    for (int i=0; i < data.length; i++) {
        String value = data[i][0];
        String expected = data[i][1];
        assertEquals(expected, FizzBuzz.compute(Integer.parseInt(value)));
    }
}
```

The diagram illustrates the execution flow of the test loop. It shows the mapping between the variables in the Java code and the elements in the `data` matrix. The `value` variable is mapped to the first element of each inner array, and the `expected` variable is mapped to the second element. The arrays are indexed as follows: 1, 2, 3, 4, 5, 5, 7. The corresponding FizzBuzz outputs are: 1, 2, Fizz, 4, Buzz, Fizz, 7. An ellipsis '...' is shown below 7.

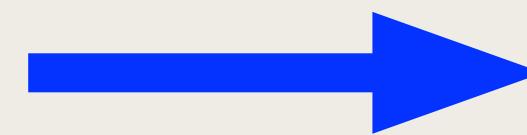
1	1
2	2
3	Fizz
4	4
5	Buzz
6	Fizz
7	7
	...



# But wait ... JUnit to the rescue

- JUnit provides @ParameterizedTest
- Run a test multiple times and provide different parameter values

1	1
2	2
3	Fizz
4	4
5	Buzz
6	Fizz
7	7
	...



**FizzBuzzTest**

**Behind the scenes, JUnit will run the test multiple times and supply the data**

**JUnit does the looping for you :-)**

# Source of Values

- When using a @ParameterizedTest, where can we get the values?

Annotation	Description
@ValueSource	Array of values: Strings, ints, doubles, floats etc
@CsvSource	Array of CSV String values
@CsvFileSource	CSV values read from a file
@EnumSource	Enum constant values
@MethodSource	Custom method for providing values

# ParameterizedTest - @CsvSource

```
@DisplayName("Testing with csv data")
@ParameterizedTest
@CsvSource({
    "1,1",
    "2,2",
    "3,Fizz",
    "4,4",
    "5,Buzz",
    "6,Fizz",
    "7,7"
})
@Order(6)
void testCsvData(int value, String expected) {
    assertEquals(expected, FizzBuzz.compute(value));
}
```

expected

Use @ParameterizedTest  
instead of @Test

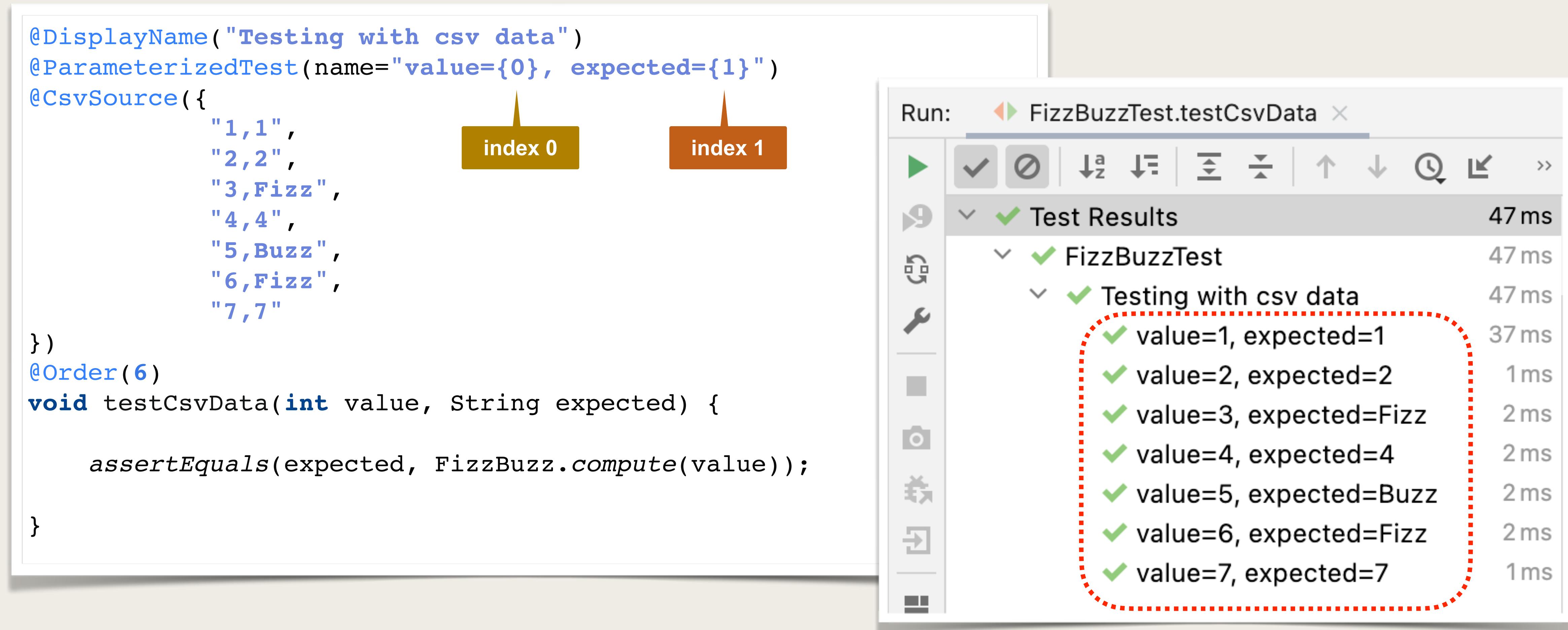
Behind the scenes, JUnit will run the  
test multiple times and  
supply the data for the parameters

(it does the looping for you :-)

	Run:	FizzBuzzTest.testCsvData	X
	▶	53 ms	
	✓	53 ms	
	✖	53 ms	
	↓ <sup>a</sup>	43 ms	
	↓ <sup>f</sup>	2 ms	
	☰	2 ms	
	÷	2 ms	
	↑	1 ms	
	↓	1 ms	
	⌚		
	⟳		
	››		
1	✓ [2] 2, 2		
2	✓ [3] 3, Fizz		
3	✓ [4] 4, 4		
4	✓ [5] 5, Buzz		
5	✓ [6] 6, Fizz		
6	✓ [7] 7, 7		
7	...		

1	1
2	2
3	Fizz
4	4
5	Buzz
6	Fizz
7	7
	...

# Customize Invocation Names



```
@DisplayName("Testing with csv data")
@ParameterizedTest(name="value={0}, expected={1}")
@CsvSource({
    "1,1",
    "2,2",
    "3,Fizz",
    "4,4",
    "5,Buzz",
    "6,Fizz",
    "7,7"
})
@Order(6)
void testCsvData(int value, String expected) {
    assertEquals(expected, FizzBuzz.compute(value));
}
```

The code above demonstrates parameterized testing with CSV data. Annotations include `@DisplayName`, `@ParameterizedTest`, `@CsvSource`, and `@Order`. The CSV source provides pairs of `value` and `expected` strings. Arrows point from the annotations to the respective CSV index values: `index 0` for the first row and `index 1` for the second.

The screenshot shows the IntelliJ IDEA Run tool window with the following details:

- Run configuration: `FizzBuzzTest.testCsvData`
- Test Results:
  - `FizzBuzzTest`: 47 ms
  - `Testing with csv data`: 47 ms
    - `value=1, expected=1`: 37 ms
    - `value=2, expected=2`: 1 ms
    - `value=3, expected=Fizz`: 2 ms
    - `value=4, expected=4`: 2 ms
    - `value=5, expected=Buzz`: 2 ms
    - `value=6, expected=Fizz`: 2 ms
    - `value=7, expected=7`: 1 ms

# Read a CSV file

The screenshot shows an IDE interface with three main components:

- Code Editor:** Displays a Java test class named `FizzBuzzTest`. It includes annotations for DisplayName, ParameterizedTest, CsvFileSource, and Order. A callout box points to the `value` parameter in the `@ParameterizedTest` annotation, with the text "reference the CSV file".
- CSV File Preview:** A modal window shows the contents of the CSV file `small-test-data.csv`, which contains the following data:

value	expected
1	1
2	2
3	Fizz
4	4
5	Buzz
6	Fizz
7	7
- Run Window:** Shows the test results for `FizzBuzzTest.testSmallDataFile`. The results table is as follows:

Test	Time
FizzBuzzTest	132 ms
Testing with Small data file	132 ms
value=1, expected=1	117 ms
value=2, expected=2	2 ms
value=3, expected=Fizz	3 ms
value=4, expected=4	6 ms
value=5, expected=Buzz	2 ms
value=6, expected=Fizz	1 ms
value=7, expected=7	1 ms

# JUnit User Guide

- Additional features for @ParameterizedTest
  - @MethodSource
  - Argument Aggregation
  - ...

<https://junit.org/junit5/docs/current/user-guide>

See section on Parameterized Tests