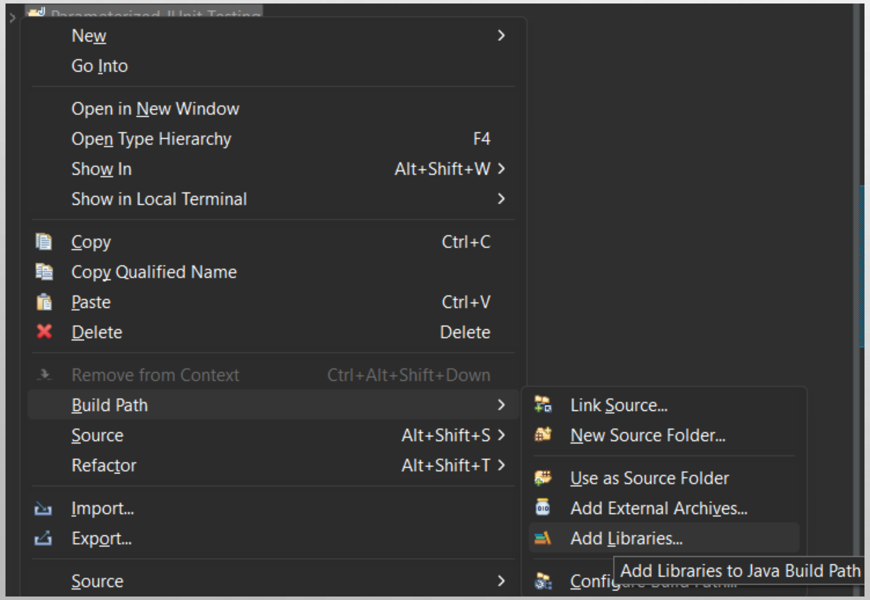
PARAMETERIZED JUNIT TESTING

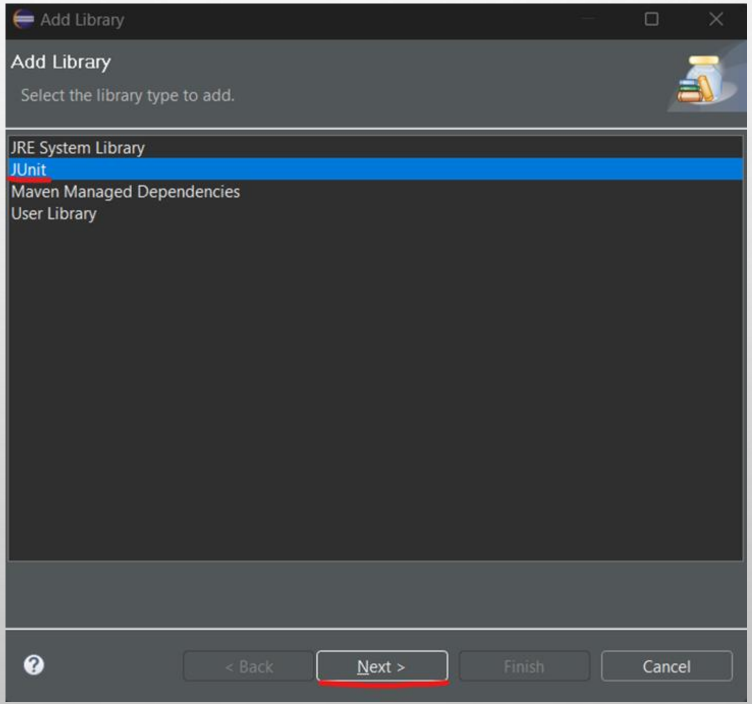
RICARDO JARDINEZ

CREATE A JAVA PROJECT AND ADD JUNIT5 LIBRARY

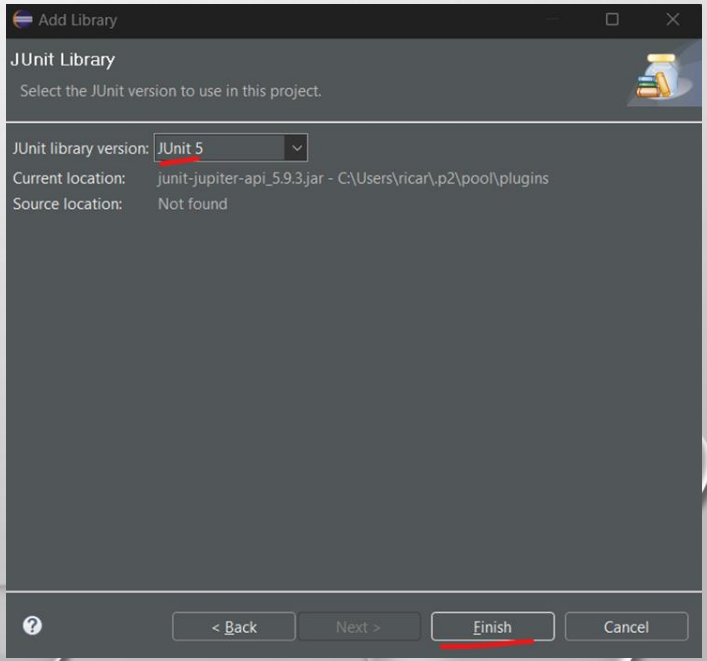
- CREATE AND CALL THE JAVA PROJECT “Parameterized JUnit Testing”
- ADD THE JUNIT LIBRARY
  - RIGHT CLICK THE JAVA PROJECT WE JUST MADE (“Parameterized JUnit Testing”)
  - CLICK ON BUILD PATH → ADD LIBRARIES
  - SELECT JUNIT FROM THE POP UP WINDOW → NEXT
  - SELECT JUNIT5 → FINISH



→



→



Add Library

JUnit Library

Select the JUnit version to use in this project.

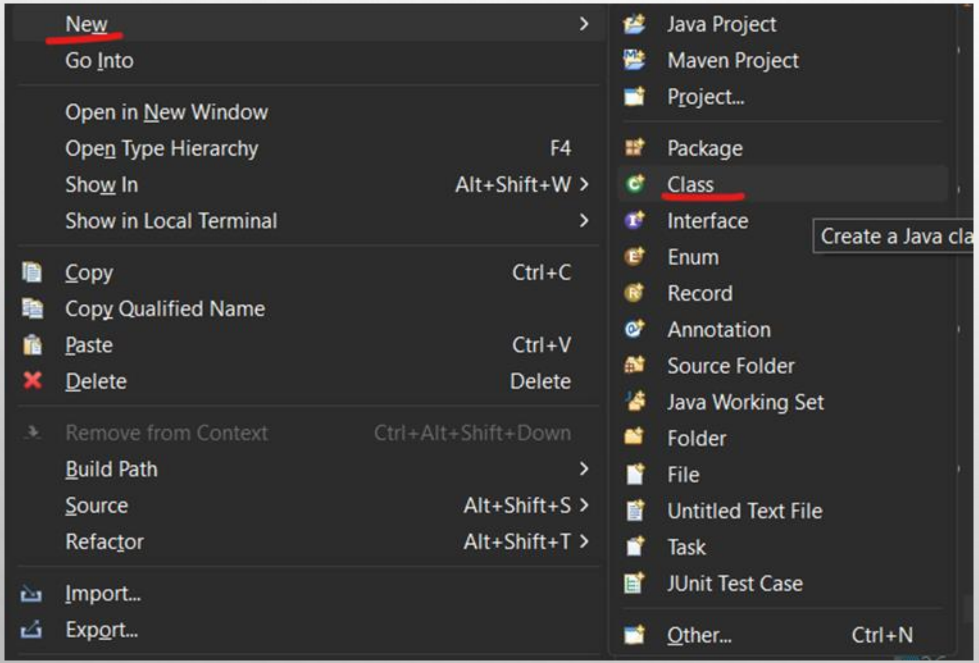
JUnit library version: JUnit 5

Current location: junit-jupiter-api_5.9.3.jar - C:\Users\ricar\.p2\pool\plugins

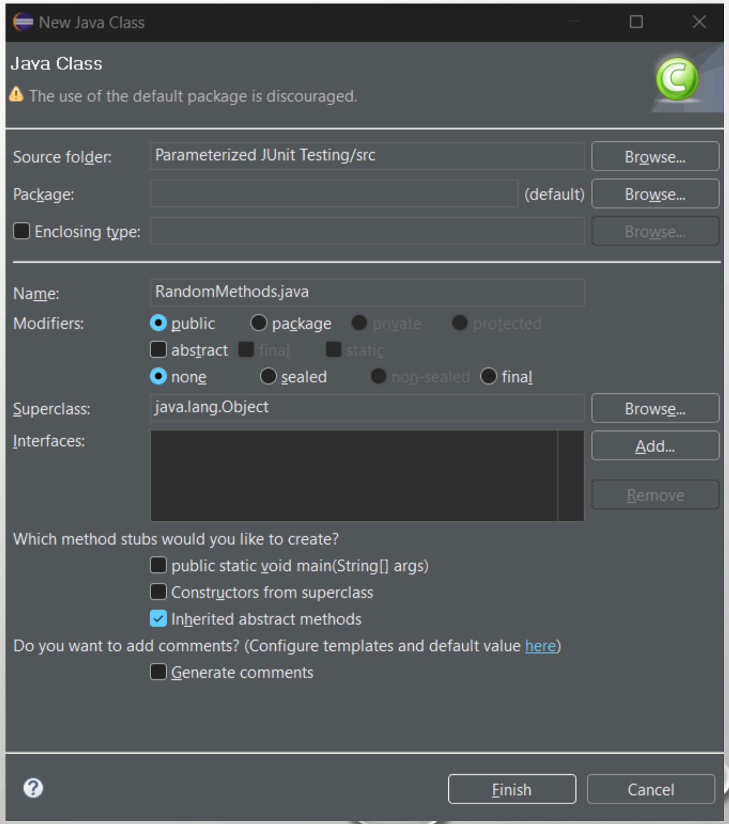
Source location: Not found

CREATE JAVA CLASS

- RIGHT CLICK ON OUR JAVA PROJECT → NEW → CLASS
- NAME JAVA CLASS “RandomMethods. JAVA ”
- CLICK FINISH



→



New Java Class

Java Class

The use of the default package is discouraged.

Source folder: Parameterized JUnit Testing/src [Browse...]

Package: (default) [Browse...]

Enclosing type: [Browse...]

Name: RandomMethods.java

Modifiers: [✓ public] package private protected

abstract final static

[✓ none] sealed non-sealed final

Superclass: java.lang.Object [Browse...]

Interfaces: [Add...] [Remove]

Which method stubs would you like to create?

public static void main(String[] args)

Constructors from superclass

[✓ Inherited abstract methods]

Do you want to add comments? (Configure templates and default value here)

Generate comments

[Finish] [Cancel]

RANDOM METHODS CODE

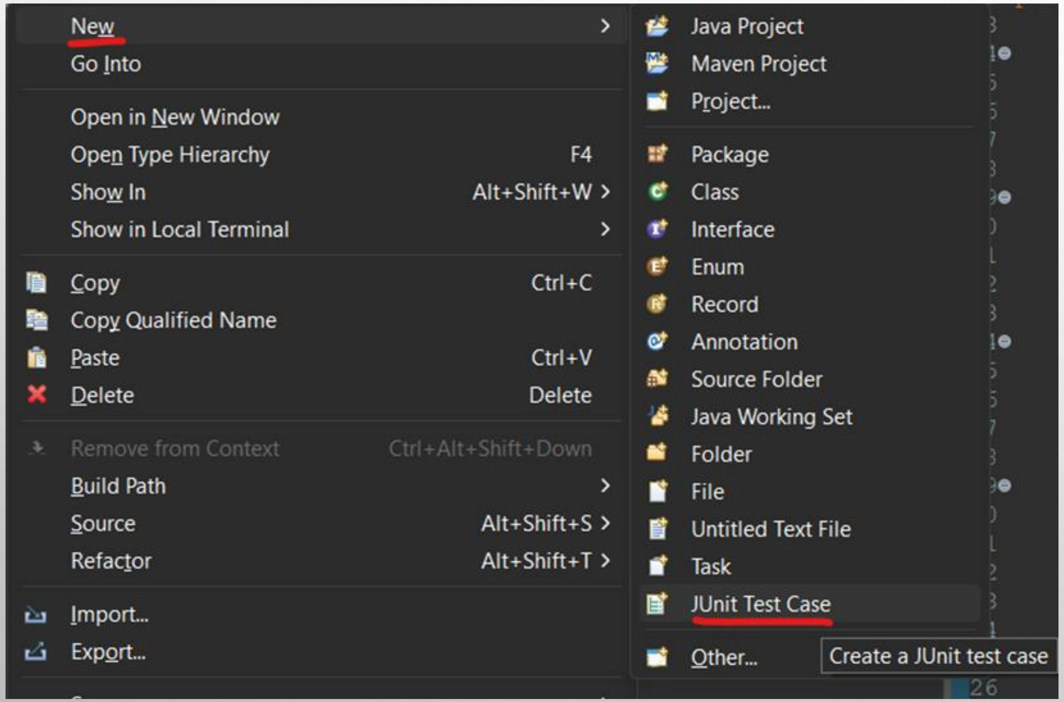
- ADD THE FOLLOWING METHODS INTO THE RandomMethods.JAVA CLASS

```
public static boolean isEven(int num)
{
        if (num %2 == 0)
                return true;
        else
                return false;
}

public static String Uppercase(String s)
{
        return s.toUpperCase();
}
```

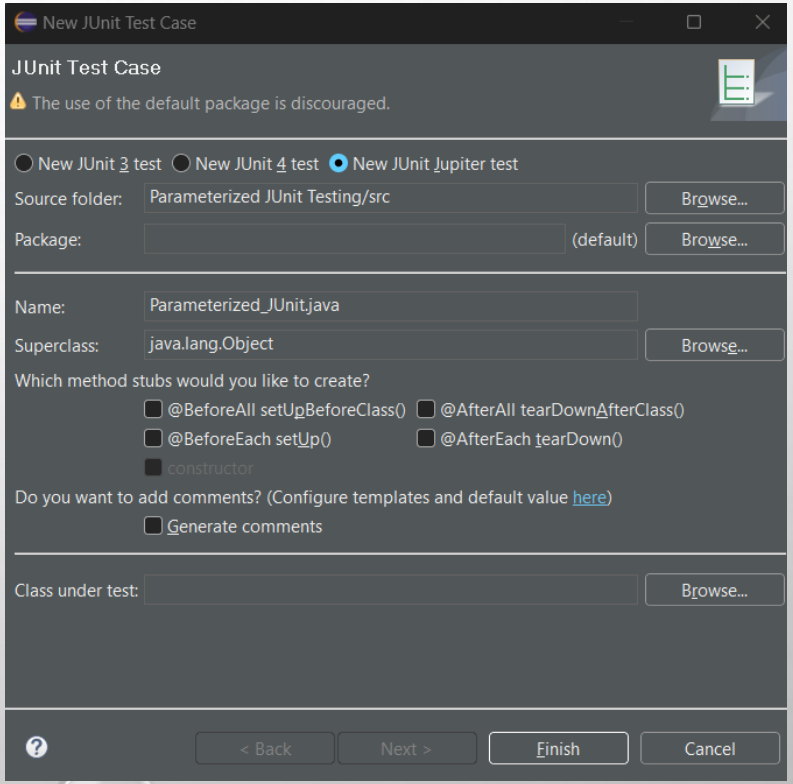
```
public static int Add(int x, int y)
{
        return x + y;
}

public static String FullName(String first, String last)
{
        return first + " " + last;
}
```

CREATE JUNIT TEST CASE

- RIGHT CLICK ON JAVA PROJECT → NEW → JUNIT TEST CASE
- NAME IT “Parameterized_JUnit.JAVA” → CLICK FINISH



→



New JUnit Test Case

JUnit Test Case

The use of the default package is discouraged.

New JUnit 3 test New JUnit 4 test [✓ New JUnit Jupiter test]

Source folder: Parameterized JUnit Testing/src Browse...

Package: (default) Browse...

Name: Parameterized_JUnit.java

Superclass: java.lang.Object Browse...

Which method stubs would you like to create?

☐ @BeforeAll setUpBeforeClass() ☐ @AfterAll tearDownAfterClass()

☐ @BeforeEach setUp() ☐ @AfterEach tearDown()

☐ constructor

Do you want to add comments? (Configure templates and default value here)

☐ Generate comments

Class under test: Browse...

< Back Next > Finish Cancel

WHAT IS PARAMETERIZED TESTING?

- PARAMETERIZED TESTING MAKES IT POSSIBLE TO RUN THE SAME TEST MULTIPLE TIMES WITH DIFFERENT ARGUMENTS
  - ALLOWS US TO QUICKLY VERIFY VARIOUS CONDITIONS WITHOUT HAVING TO WRITE A TEST FOR EACH CASE
- NEED TO USE THE @ParamerterizedTest ANNOTATION
  - NEED TO IMPORT “org.junit.jupiter.params.ParameterizedTest” WHEN DOING PARAMETERIZED TESTS
- NEED TO DECLARE AN ARGUMENT SOURCE FOR THE TEST
  - DECLARE THESE ARGUMENTS SOURCES WITH DIFFERENT ARGUMENT SOURCE ANNOTATIONS
  - @ValueSoruce @NullSoruce @MethodSource @ArgumentSource @CsvSource

@ValueSource

- SIMPLEST ARGUMENT SOURCE
- WE CAN PASS AN ARRAY OF LITERAL VALUES TO THE TEST METHOD
  - CAN SPECIFY AN ARRAY OF TYPES int, long, float, double, char, boolean, string, short, byte
- JUNIT WILL RUN THE TEST N NUMBER OF TIMES, EACH TIME ASSIGNS ONE ARGUMENT FROM THE ARRAY TO THE METHOD PARAMETER
- NEED TO IMPORT “org.junit.jupiter.params.provider.ValueSource”
- WHEN TO USE @ValueSource:
  - WHEN YOU HAVE A SINGLE DATA ENTRY
  - WHEN ITS POSSIBLE TO APPLY THE BOUNDARY VALUE ANALYSIS

@ValueSource TEST CASE

- TESTING @VALUESOURCE WITH "IsEven" METHODS.

```
@ParameterizedTest
@ValueSource (ints = {2, 4, 6, 8, 10})
void testEven(int nums)
{
	assertEquals(true, RandomMethods.isEven(nums));
}

OR

@ParameterizedTest
@ValueSource (ints = {2, 4, 6, 8, 10})
void testEven2(int nums)
{
	//uses assertTrue instead of assertEquals
	assertTrue(RandomMethods.isEven(nums));
}
```

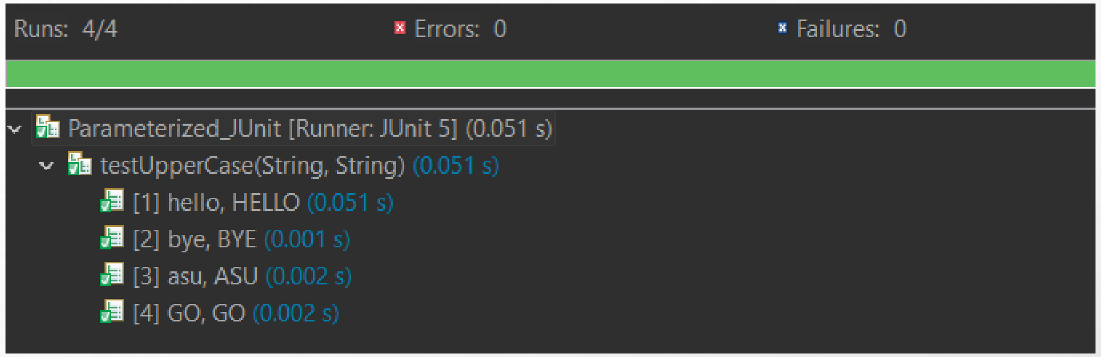
@CsvSource

- ALLOWS IS TO USE A LIST/ARRAY OF COMMA-SEPARATED STRING VALUES
    - EACH ARRAY ENTRY CORRESPONDS TO A LINE IN A CSV FILE
- THIS SOURCE TAKES ONE ARRAY ENTRY EACH TIME, SPLITS IT BY THE COMMA AND PASSES EACH ARRAY TO THE ANNOTATED TEST METHOD AS SEPARATE PARAMETERS
- COMMA IS THE DEFAULT SEPARATOR BUT WE CAN CHANGE IT USING THE delimiter ATTRIBUTE
- THE @CsvSource ANNOTATION MAKES IT POSSIBLE TO PROVIDE MULTIPLE PARAMETERS TO THE TEST METHOD IN A COMPACT WAY
- NEED TO IMPORT “org.junit.jupiter.params.provider.CsvSource”

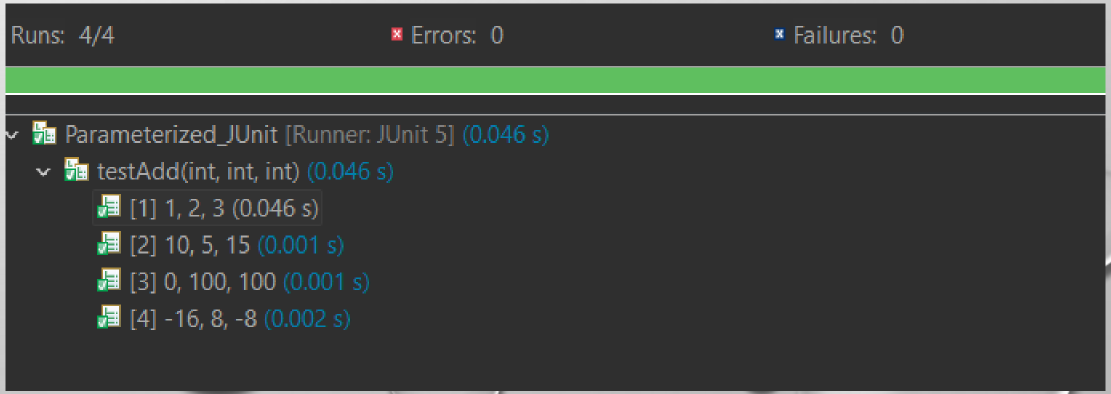
@CsvSource TEST CASE

- TESTING @CsvSource WITH "Uppercase" AND "Add" METHODS

```
@ParameterizedTest
@CsvSource ({
	"hello,HELLO", "bye,BYE", "asu,ASU", "GO,GO"
	})
OR
@ParameterizedTest
@CsvSource (value = {"hello~HELLO", "bye~BYE", "asu~ASU", "GO~GO"}, delimiter = '~')
void testUpperCase(String n1, String n2)
{
	assertEquals(n2, RandomMethods.Uppercase(n1));
}

@ParameterizedTest
@CsvSource ({
	"1,2,3", "10,5,15", "0,100,100", "-16,8,-8"
	})
void testAdd(int n1, int n2, int answer)
{
	assertEquals(answer, RandomMethods.Add(n1, n2));
```



```
Runs: 4/4        Errors: 0        Failures: 0

Parameterized_JUnit [Runner: JUnit 5] (0.051 s)
  testUpperCase(String, String) (0.051 s)
    [1] hello, HELLO (0.051 s)
    [2] bye, BYE (0.001 s)
    [3] asu, ASU (0.002 s)
    [4] GO, GO (0.002 s)
```

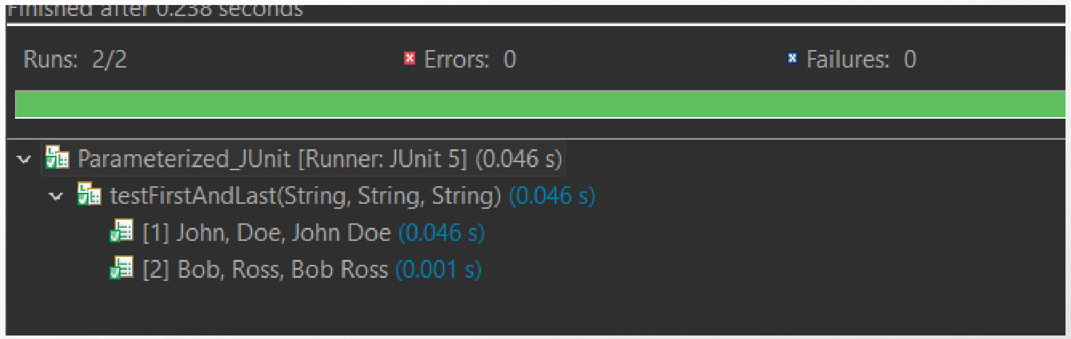


```
Runs: 4/4        Errors: 0        Failures: 0

Parameterized_JUnit [Runner: JUnit 5] (0.046 s)
  testAdd(int, int, int) (0.046 s)
    [1] 1, 2, 3 (0.046 s)
    [2] 10, 5, 15 (0.001 s)
    [3] 0, 100, 100 (0.001 s)
    [4] -16, 8, -8 (0.002 s)
```

@CsvSource TEST CASE CONT

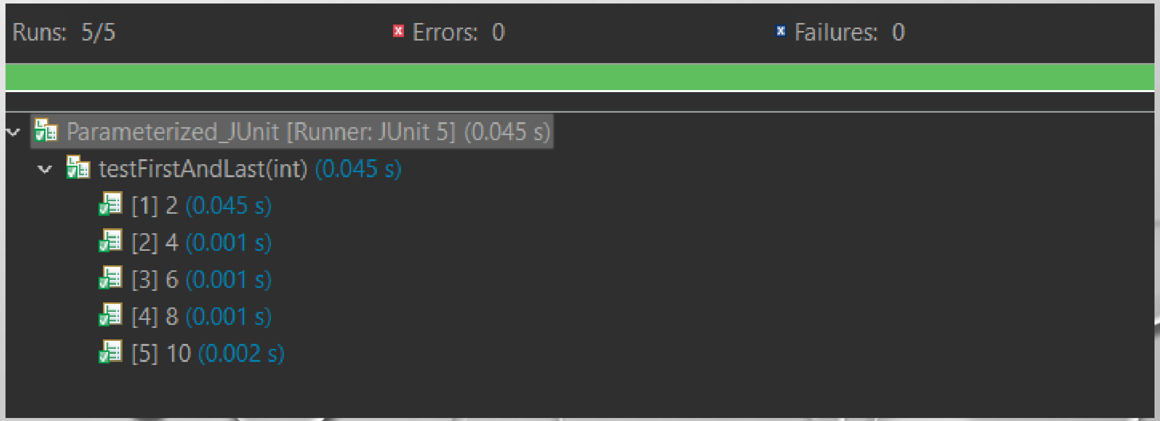
- TESTING @CsvSource WITH “FullName” METHODS

```
@ParameterizedTest
@CsvSource ({
        "John, Doe, John Doe", "Bob, Ross, Bob Ross"
})
void testFirstAndLast(String first, String last, String FullName)
{
        assertEquals(FullName, RandomMethods.FullName(first, last));
}
```



```
Finished after 0.238 seconds
Runs:  2/2                Errors:  0                Failures:  0

⌄ Parameterized_JUnit [Runner: JUnit 5] (0.046 s)
   ⌄ testFirstAndLast(String, String, String) (0.046 s)
        [1] John, Doe, John Doe (0.046 s)
        [2] Bob, Ross, Bob Ross (0.001 s)
```

```
@ParameterizedTest
@CsvSource ({
                "2", "4", "6", "8", "10"
})
void testFirstAndLast(int num1)
{
        assertEquals(true, RandomMethods.isEven(num1));
}
```



```
Runs:  5/5                Errors:  0                Failures:  0

⌄ Parameterized_JUnit [Runner: JUnit 5] (0.045 s)
   ⌄ testFirstAndLast(int) (0.045 s)
        [1] 2 (0.045 s)
        [2] 4 (0.001 s)
        [3] 6 (0.001 s)
        [4] 8 (0.001 s)
        [5] 10 (0.002 s)
```

@MethodSoruce

- APPROACH TO PROVIDING MORE COMPLEX ARGUMENTS
- @MethodSoruce ALLOWS IS TO REFER TO A FACTORY METHOD THAT RETURNS THE ARGUMENTS
- CREATING AND USING OBJECTS
- NEED TO IMPORT “org.junit.jupiter.params.provider.MethodSource”
- THE NAME WE GIVE TO @MethodSoruce NEEDS TO MATCH AN EXISTING METHOD
  - WHEN WE DON’T PROVIDE @MethodSoruce WITH A NAME FOR THE JUNIT WILL SEARCH FOR A SOURCE METHOD WITH THE SAME NAME AS THE TEST METHOD

@MethodSoruce TEST CASE

- TESTING @MethodSource WITH “Add” AND “Uppercase” METHODS

```
private static Object[] inputs()
{
	return new Object[][]
		{
		{1,2,3},
		{10,5,15},
		{0,100,100},
		{-16,8,-8}
		};
}
@ParameterizedTest
@MethodSource("inputs")
void AddMethodSource (int num1, int num2, int answer)
{
	assertEquals(answer, RandomMethods.Add(num1, num2));

}
```

```
private static Object[] inputs()
{
	return new Object[][]
		{
		{"hello","HELLO"},
		{"bye","BYE"},
		{"asu","ASU"},
		{"GO","GO"}
		};
}
@ParameterizedTest
@MethodSource("inputs")
void UpperCaseMethodTesting (String users, String Answer)
{
	assertEquals(Answer,RandomMethods.Uppercase(users));

}
```

@MethodSoruce TEST CASE CONT

- TESTING @MethodSource WITH “isEven” AND “FullName” METHODS

```
private static Object[] inputs()
{
        return new Object[][]
                {
                {2},
                {4},
                {6},
                {8},
                {10}
                };
}
@ParameterizedTest
@MethodSource("inputs")
void isEvenMethodTesting (int num1)
{
        assertEquals(true, RandomMethods.isEven(num1));
}
```

```
private static Object[] inputs()
{
        return new Object[][]
                {
                {"John", "Doe", "John Doe"},
                {"Bob", "Ross", "Bob Ross"}
                };
}
@ParameterizedTest
@MethodSource("inputs")
void AddMethodSource (String first, String last, String answer)
{
        assertEquals(answer, RandomMethods.FullName(first, last));

}
```