What is Parameterized Test in Junit?

Parameterized test is to execute the same test over and over again using different values. It helps developer to save time in executing same test which differs only in their inputs and expected results.

Using Parameterized test, one can set up a test method that retrieves data from some data source.

Consider a simple test to sum different numbers. The code may look like -

```
11⊖    @Test
12     public void testAirthematicTest() {
13         // assert statements
14         assertEquals("10 +10 must be 20", 20, airthematic.sum(10, 10));
15         assertEquals("20 +20 must be 40", 40, airthematic.sum(20, 20));
16         assertEquals("30 +10 must be 40", 40, airthematic.sum(30, 10));
17
18     }
19 }
20
```

The approach above leads to lot of redundancy.

We need a simple approach and. Using parameterized test you can just add a method to input 10 data inputs and your test will run 10 times automatically.

Steps to create a Parameterized JUnit test

Following code shows an example for a parameterized test. It tests sum() method of the Arithmetic class :

Step 1) Create a class. In this example, we are going to input two numbers by using sum (int,int) method which will return the sum of given numbers

```
1  package junitTutorial;
2
3  public class Airthematic {
4⊖     public int sum(int a,int b){
5          return a+b;
6      }
7
8  }
```

Step 2) Create a parameterized test class

```
11 @RunWith(Parameterized.class)
12 public class AirthematicTest {
13     private int firstNumber;
14     private int secondNumber;
15     private int expectedResult;
16     private Airthematic airthematic;
17
```

Code Explanation

- **Code Line 11:** Annotate your test class using @runWith(Parameterized.class).
- **Code Line 13:** Declaring the variable 'firstNumber' as private and type as int.
- **Code Line 14:** Declaring the variable 'secondNumber'as private and type as int.
- **Code Line 15:** Declaring the variable 'expectedResult'as private and type as int.
- **Code Line 16:** Declaring the variable 'airthematic'as private and type as Airthematic.

@**RunWith**(class_name.class): @**RunWith** annotation is used to specify its runner class name. If we don't specify any type as a parameter, the runtime will choose **BlockJunit4ClassRunner** by default.

This class is responsible for tests to run with a new test instance. It is responsible for invoking JUnit lifecycle methods such as setup(associate resources) and teardown(release resources).

To parameterize you need to annotate using @RunWith and pass required .class to be tested

Step 3) Create a constructor that stores the test data. It stores 3 variables

```
17
18     public AirthematicTest(int firstNumber, int secondNumber, int expectedResult) {
19         super();
20         this.firstNumber = firstNumber;
21         this.secondNumber = secondNumber;
22         this.expectedResult = expectedResult;
23     }
24
```

Step 4) Create a static method that generates and returns test data.

```
30    @Parameterized.Parameters
31    public static Collection input() {
32        return Arrays.asList(new Object[][] { { 1, 2, 3 }, { 11, 22, 33 },
33                { 111, 222, 333 }, { 10, 9, 19 }, { 100, 9, 109 } });
34    }
35
```

Code Line 32,33: Creating a two-dimensional array (providing input parameters for addition). Using asList method we convert the data into a List type. Since, the return type of method input is collection.

Code Line 30: Using **@Parameters** annotation to create a set of input data to run our test.

The static method identified by @Parameters annotation returns a Collection where each entry in the Collection will be the input data for one iteration of the test.

Consider the elemenent

{1,2,3}

Here

firstNumber =1

secondNumber=2

expectedResult=3

Here each array element will be passed to the constructor, one at a time as the class is instantiated multiple times.

Step 5) The complete code

```
1  package junitTutorial;
2
3  import static org.junit.Assert.assertEquals;
4  import java.util.Arrays;
5  import java.util.Collection;
6  import org.junit.Before;
7  import org.junit.Test;
8  import org.junit.runner.RunWith;
9  import org.junit.runners.Parameterized;
10
11 @RunWith(Parameterized.class)
12 public class AirthematicTest {
13     private int firstNumber;
14     private int secondNumber;
15     private int expectedResult;
16     private Airthematic airthematic;
17
18     public AirthematicTest(int firstNumber, int secondNumber, int expectedResult) {
19         super();
20         this.firstNumber = firstNumber;
21         this.secondNumber = secondNumber;
22         this.expectedResult = expectedResult;
23     }
24
25     @Before
26     public void initialize() {
27         airthematic = new Airthematic();
28     }
29
30     @Parameterized.Parameters
31     public static Collection input() {
32         return Arrays.asList(new Object[][] { { 1, 2, 3 }, { 11, 22, 33 },
33                 { 111, 222, 333 }, { 10, 9, 19 }, { 100, 9, 109 } });
34     }
35
36     @Test
37     public void testAirthematicTest() {
38         System.out.println("Sum of Numbers = : " + expectedResult);
39         assertEquals(expectedResult, airthematic.sum(firstNumber, secondNumber));
40     }
```

Code Explanation:

- **Code Line 25:** Using @Before annotation to setup the resources (Airthematic.class here). The @Before annotation is used here to run before each test case. It contains precondition of the test.
- **Code Line 36:** Using @Test annotation to create our test.
- **Code Line 39:** Creating an assert statement to check whether our sum is equivalent to what we expected.

Step 6) Create a test runner class to run parameterized test:

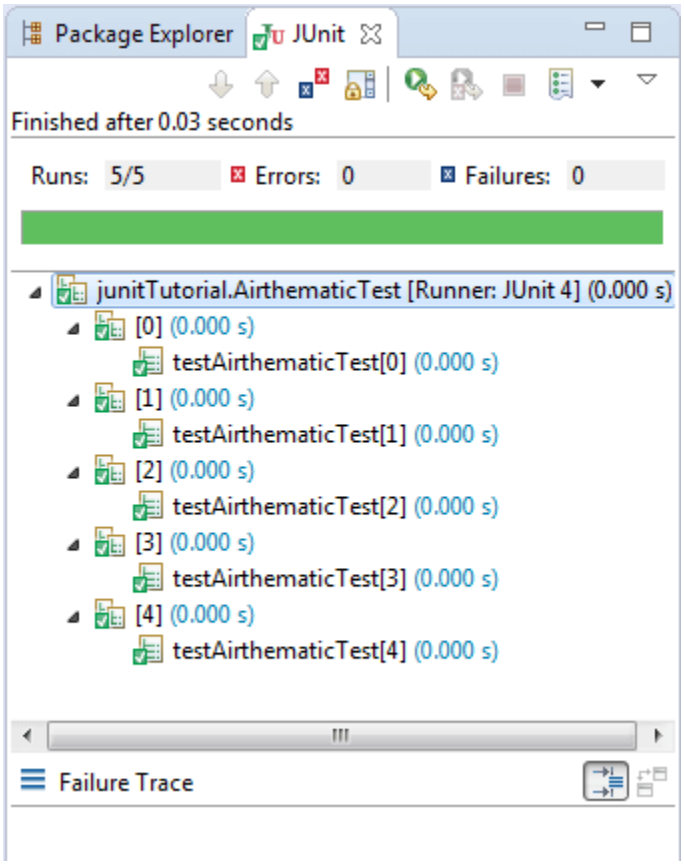
```
1  package junitTutorial;
2
3  import org.junit.runner.JUnitCore;
4  import org.junit.runner.Result;
5  import org.junit.runner.notification.Failure;
6
7  public class Test {
8     public static void main(String[] args) {
9         Result result = JUnitCore.runClasses(AirthematicTest.class);
10        for (Failure failure : result.getFailures()) {
11           System.out.println(failure.toString());
12        }
13        System.out.println(result.wasSuccessful());
14    }
15 }
```

Code Explanation:

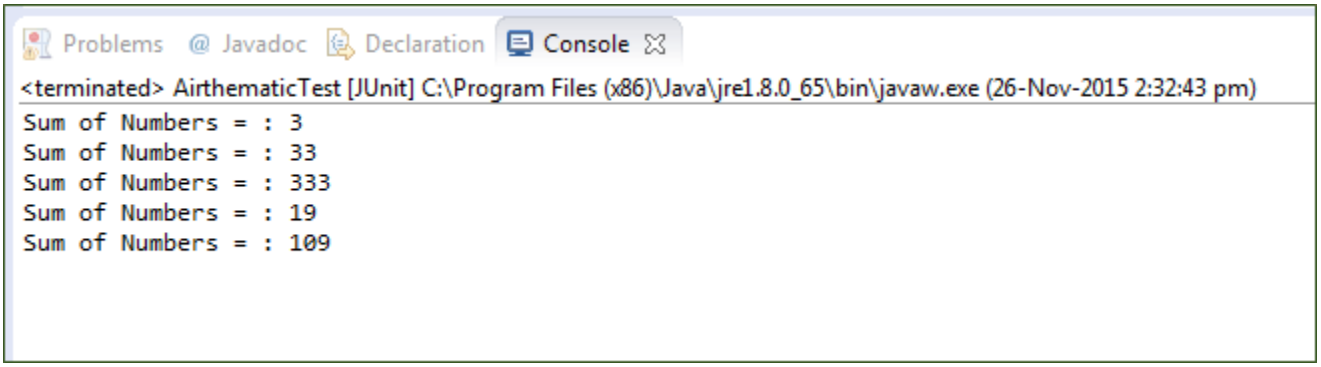
- **Code Line 8:** Declaring the main method of the class test which will run our JUnit test.
- **Code Line 9:** Executing test cases using JunitCore.runclasses, it will take the test class name as a parameter (In our example we are using Airthematic.class).
- **Code Line 11:** Processing the result using for loop and printing out failed result.
- **Code Line 13:** Printing out the successful result.

Output:

Here is the output which shows successful test with no failure trace as given below:



See the result on console,which shows addition of two numbers :-



Summary:

Parameterized test enables developer to execute the same test over and over again using different values.

Important annotations to be used during parameterization

- **@RunWith**
- **@Parameters**