 Parameterized test with value source

**package** junitTestScripts;

**import** org.junit.jupiter.params.ParameterizedTest;

**import** org.junit.jupiter.params.provider.ValueSource;

**public** **class** ParameterizedValueSource {

// passing test data to the test method

// to pass test data to a Junit test cases is using annotation @Valuesource

@ParameterizedTest(name= "{index} - Run the test")

@ValueSource(ints = {11,12,13,14,15})

**public** **void** valuesourcedemo(**int** value)

{

System.***out***.println(" the integer values" + value);

}

@parameterizedtest name= "{index} - Run the test for string value")

@ValueSource(strings = {"abc","sonal","selenium"})

**public** **void** valuesourcedemoString(String value)

{

System.***out***.println(" the String values" + value);

}

}

=======================================================================

Enum -> Class & datatype

final String str1 = “selenium”

enum str1 = “SELENIUM”,”TOOL:,”JMETER”

The value in enum type are constant and are represented in UPPER CASE

Whenever we want a variable to have a specific set of values that are final and constant-> then use enum

Junit supports sending **enum values** to your Test method using **@EnumSource**

\===========\\

**package** junitTestScripts;

**import** org.junit.jupiter.params.ParameterizedTest;

**import** org.junit.jupiter.params.provider.EnumSource;

**public** **class** ParametrizedEnumSource {

String str = "Selenium";

// values = variable name of type enum

// Values is an Enum object

**enum** Values{

***SELENIUM***, ***JMETER***, ***POSTMAN***

}

@ParameterizedTest(name = "Enum value {arguments}") // stores the values {SELENIUM, JMETER, POSTMAN}

@EnumSource(Values.**class**)

**public** **void** TestParametrizedEnumSource(Object value)

{

System.***out***.println(value.toString());

}

}

=====================

Data type : Stream : List of values of any data type

Using  a user defined method which return a stream of value, Junit can pass these stream of values to you test method

> create a user defined method which stores a list of values

> This user defined method will be of type Stream, return stream of values

> we use the annotation @ParameterizedTest and @MethodSource to send stream of data to test method.

/=============//

**package** junitTestScripts;

**import** java.util.stream.Stream;

**import** org.junit.jupiter.params.ParameterizedTest;

**import** org.junit.jupiter.params.provider.MethodSource;

**public** **class** ParametrizedTestMethodSource {

**public** **static** Stream<String> stringParamters()

{

**return** Stream.*of*("Monday","Tuesday","Wednesday");

}

**public** **static** Stream<Integer> intParamters()

{

**return** Stream.*of*(100,200,300);

}

@ParameterizedTest(name = "Method value {arguments}")

@MethodSource("stringParamters")

**public** **void** getdatafromMethods(String input)

{

System.***out***.println("The value from String method is : " + input);

// System.out.println("The value from integer method is : " + input1);

}

@ParameterizedTest(name = "Method value {arguments}")

@MethodSource("intParamters")

**public** **void** getdatafromIntMethods(**int** input)

{

System.***out***.println("The value from Integer method is : " + input);

}

}

=============================================

@CSVSource

Way of store the data in the form of rows and columns

Simple way of storing the data

Example:

Toolname version

Selenium 4

Postman 2

Jmeter 5

//=============================//

**package** junitTestScripts;

**import** org.junit.jupiter.params.ParameterizedTest;

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

**public** **class** ParameterizedCSVsource {

@ParameterizedTest(name = "CVS source {arguments}")

@CsvSource({

"Selenium , v4",

"Junit , v5",

"TestNG , v7",

"JMeter , v5",

"Postman , v2",

})

**public** **void** getdatafromCSV(String value1, String value2)

{

System.***out***.println(value1 + " : " + value2);

}

}