ABOUT VINOD KUMAR KASHYAP

Vinod is Sun Certified and love to work in Java and related technologies. Having more than 10 years of experience, he had developed software's including technologies like Java, Hibernate, Struts, Spring, HTML 5, jQuery, CSS, Web Services, MongoDB, AngularJS. He is also a JUG Leader of Chandigarh Java User Group.

JUnit FixMethodOrder Example

Posted by: Vinod Kumar Kashyap in junit February 10th, 2017

With this example, we are going to demonstrate users when, how and why JUnit

```
FixMethodOrder
```

annotation is used. In previous example JUnit Hello World, users have seen how they can start using JUnit. Users are advised to see the setup of project in JUnit Hello World example, if they want to continue with Maven.

This example is useful in cases where user wants to run their test cases in particular order. Users are required to have basic knowledge of Java for this example. We will follow with an short example to show the process of using JUnit

```
FixMethodOrder
```

annotation.

Tip
You may skip introduction and jump directly to the **beginning of the example** below.

1. Tools Used

For this example you will need:

- Java 8
- JUnit 4.12

2. Introduction

JUnit

```
@FixMethodOrder
```

annotation is used with JUnit for specifying order of the methods to run. JUnit is a testing framework for Java. Users who are not aware of the JUnit, can refer to the post JUnit Hello World.

By default there is no specific order of execution and the test cases run without any predictability.

```
@FixMethodOrder
```

is useful in instances, where users need to run their test cases in order of the names of the test cases.

```
@FixMethodOrder
```

annotation helps to achieve this goal.

3. JUnit FixMethodOrder Annotation

Furthermore this annotation makes use of

```
MethodSorters
```

Enum as parameter name to identify the order. In order to start with

```
@FixMethodOrder
```

annotation, let's do a quick look into the

```
MethodSorters
```

Enum.

3.1 MethodSorters Enum

```
MethodSorters
```

Enum contains 3 types of constants.

- **DEFAULT**: Default implementation and the order is not predictable.
- **JVM**: This constant leaves the execution of order on JVM.
- **NAME_ASCENDING**: This is mostly used constant that sorts the method name in ascending order.

```
MethodSorters.NAME_ASCENDING
```

is the most noteworthy, and especially relevant for users. It uses

```
Method.toString()
```

method, in case there is a tie breaker (i.e. method with same name) between the method names.

Let's start with an example.

4. Example

First of all let's create a class without the

```
@FixMethodOrder
```

annotation. This is a simple class with 3 test cases:

- firstTest()
- secondTest()
- thirdTest()

These are simple test cases which prints out the name of the test case. The output result is also shown after class.

JUnitFixMethodOrderTest

```
package junit;

import org.junit.Test;

public class JUnitFixMethodOrderTest {

```

```
    @Test
    public void firstTest() {
        System.out.println("First Test");
    }

    @Test
    public void thirdTest() {
        System.out.println("Third Test");
    }

    @Test
    public void secondTest() {
        System.out.println("Second Test");
    }
}
```

The output result is shown below:

```
Third Test
First Test
Second Test
```

Users can see, that the order of execution is not predictable at all, rather test cases run randomly. Now make changes to the class to include the

```
@FixMethodOrder
```

annotation.
See the changes below, especially the highlighted lines.

```
package junit;

import org.junit.FixMethodOrder;
import org.junit.Test;
import org.junit.runners.MethodSorters;

@FixMethodOrder(MethodSorters.NAME_ASCENDING)
public class JUnitFixMethodOrderTest {

    @Test
    public void firstTest() {
        System.out.println("First Test");
    }

    @Test
    public void thirdTest() {
        System.out.println("Third Test");
    }

    @Test
    public void secondTest() {
        System.out.println("Second Test");
    }
}
```

The output result is shown below:

```
First Test
Second Test
Third Test
```

Hence from the above output, it is clear that the JUnit

```
@FixMethodOrder
```

annotation helps test cases run according to the names of methods.

5. Conclusion

In this example, users have learnt about the use of the JUnit

```
@FixMethodOrder
```

annotation. Users get an insight into how, why and when they should use

```
@FixMethodOrder
```

annotation.

6. Download The Source Code

This was an example of JUnit

```
@FixMethodOrder
```

annotation.

Download
You can download the java file of this example here: **JUnitFixMethodOrderTest.zip**