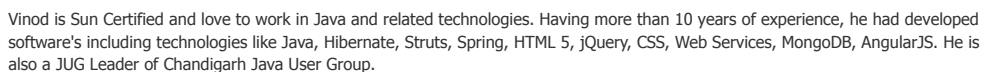




ABOUT VINOD KUMAR KASHYAP



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

FixMethodOrder

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.

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

Subscribe to our newsletter and download the JUnit Programming Cookbook [right now!](#)

In order to help you master unit testing with JUnit, we have compiled a kick-ass guide with all the major JUnit features and use cases! Besides studying them online you may download the eBook in PDF format!

Email address:

Your email address

Sign up

For this example you will need:

- Java 8
- JUnit 4.12

180,180 insiders are already enjoying weekly updates and complimentary whitepapers!


Join them now to gain **EXCLUSIVE ACCESS** to the latest news in the Java community, as well as insights about Android, Scala, Groovy and other related technologies.

Email address:

Your email address

Sign up

The image shows a yellow pencil with a pink eraser and a blue eraser, resting on a piece of paper. The paper has the 'Java Code Geeks' logo at the top, which includes a stylized 'J' and the text 'Java Code Geeks'. Below the logo, there is some handwritten text in blue ink, which appears to be a list of items or a set of instructions. The text is partially obscured by the pencil and the eraser.



With **1,240,6** unique visitors **500** authors placed among related sites and Constantly being looked for par encourage you So If you have

unique and interesting content then vo

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](#)

```
01 package junit;
02
03 import org.junit.Test;
04
05 public class JUnitFixMethodOrderTest {
06
```

```

07     @Test
08     public void firstTest() {
09         System.out.println("First Test");
10     }
11
12     @Test
13     public void thirdTest() {
14         System.out.println("Third Test");
15     }
16
17     @Test
18     public void secondTest() {
19         System.out.println("Second Test");
20     }
21 }

```

The output result is shown below:

```

1 Third Test
2 First Test
3 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.

```

01 package junit;
02
03 import org.junit.FixMethodOrder;
04 import org.junit.Test;
05 import org.junit.runners.MethodSorters;
06
07 @FixMethodOrder(MethodSorters.NAME_ASCENDING)
08 public class JUnitFixMethodOrderTest {
09
10     @Test
11     public void firstTest() {
12         System.out.println("First Test");
13     }
14
15     @Test
16     public void thirdTest() {
17         System.out.println("Third Test");
18     }
19
20     @Test
21     public void secondTest() {
22         System.out.println("Second Test");
23     }
24 }

```

The output result is shown below:

```

1 First Test
2 Second Test
3 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**