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 RunListener Example

Posted by: Vinod Kumar Kashyap in junit February 22nd, 2017

In JUnit RunListener Example, we shall show users how they can add

```
RunListener
```

to the test cases. There are cases when we want to respond to the events during a test case run. Here we can extend the

```
RunListener
```

class and override the methods according to our implementation. The JUnit

```
RunListener
```

can listen to the events of the JUnit lifecycle.

If somehow, listener throws an error, then it will be removed from remainder of the test case run. Let's see the example.

Furthermore, if user wants to learn about the basic of JUnit, they are advised to visit the below examples from Java Code Geeks.

- JUnit Hello World Example
- JUnit FixMethodOrder Example
- JUnit Keyboard Input Example
- JUnit MultiThreaded Test Example
- JUnit Group Tests Example

1. Technology Stack

We have used following technologies for this example to work.

- Java
- JUnit 4.12
- Maven – build and dependency tool
- Eclipse – IDE for writing code

2. Project Setup

Tip
You may skip project creation and jump directly to the **beginning of the example** below.
Create a new maven project
Select

```
File -> New -> Maven Project
```

Figure 1: JUnit RunListener Example setup 1

Click on Next button. Fill in the details as detailed below:

Figure 2: JUnit RunListener Example setup 2

With the click on Finish button, we are ready to start coding for this example.

3. JUnit RunListener Example

First of all we need to provide the JUnit jar to the project. For this we add following lines to the

```
pom.xml
```

pom.xml

```
<dependencies>
     <dependency>
          <groupId>junit</groupId>
          <artifactId>junit</artifactId>
          <version>4.12</version>
     </dependency>
</dependencies>
```

Now, we will create a class which will extends the

```
RunListener
```

class. This class has many methods that we can override. It is our wish which methods to implement and which to ignore.
For the sake of knowledge of users, we have taken all methods here and write about them. You can skip some of them. Code of this class is self explanatory.

OurListener.java

```
package junitrunlistener;

import org.junit.runner.Description;
import org.junit.runner.Result;
import org.junit.runner.notification.Failure;
import org.junit.runner.notification.RunListener;

public class OurListener extends RunListener {

    // Called before any tests have been run.
```

```
    public void testRunStarted(Description description) throws java.lang.Exception {
        System.out.println("Test cases to execute : " + description.testCount());
    }

    // Called when all tests have finished
    public void testRunFinished(Result result) throws java.lang.Exception {
        System.out.println("Test cases executed : " + result.getRunCount());
    }

    // Called when an atomic test is about to be started.
    public void testStarted(Description description) throws java.lang.Exception {
        System.out.println("Execution Started : " + description.getMethodName());
    }

    // Called when an atomic test has finished, whether the test succeeds or
    // fails.
    public void testFinished(Description description) throws java.lang.Exception {
        System.out.println("Execution Finished : " + description.getMethodName());
    }

    // Called when an atomic test fails.
    public void testFailure(Failure failure) throws java.lang.Exception {
        System.out.println("Execution Failure : " + failure.getException());
    }

    // Called when a test will not be run, generally because a test method is
    // annotated with Ignore.
    public void testIgnored(Description description) throws java.lang.Exception {
        System.out.println("Execution Ignored : " + description.getMethodName());
    }

    // Called when an atomic test flags that it assumes a condition that is false
    public void testAssumptionFailure(Failure failure){
        System.out.println("Assumption Failure : " + failure.getMessage());
    }
}
```

3.1. Test Classes

We will create 2 test classes for this example.

TestClassA.java

It has 2 test methods,

```
test_A_1()
```

and

```
test_A_2()
```

.

```
01 package junitrunlistener;
02
03 import static org.junit.Assert.assertTrue;
04
05 import org.junit.Test;
06
07 public class TestClassA {
08
09     @Test
10     public void test_A_1(){
11         assertTrue(1==2);
12     }
13
14     @Test
15     public void test_A_2(){
16         assertTrue(true);
17     }
18 }
```

Due to line no 11(highlighted),

```
test_A_1()
```

method fails and throws

```
java.lang.AssertionError
```

.

TestClassB.java

This class also have 2 methods,

```
test_B_1()
```

and

```
test_B_2()
```

.

```
01 package junitrunlistener;
02
```

```
03 import static org.junit.Assert.assertTrue;
04
05 import org.junit.Ignore;
06 import org.junit.Test;
07
08 public class TestClassB {
09
10     @Test
11     public void test_B_1(){
12         assertTrue(true);
13     }
14
15     @Ignore
16     @Test
17     public void test_B_2(){
18         assertTrue(2==5);
19     }
20 }
```

As you can see that

```
test_B_2()
```

is marked with

```
@Ignore
```

annotation. This annotation will simply ignore this test case from running.

3.2. Main Class

TestClassRun.java

Now, we are ready to run our tests. Create a class with the following code.

```
01 package junitrunlistener;
02
03 import org.junit.runner.JUnitCore;
04
05 public class TestClassRun {
06
07     public static void main(String[] args) {
08         JUnitCore runner = new JUnitCore();
09         runner.addListener(new OurListener());
10         runner.run(TestClassA.class, TestClassB.class);
11     }
12 }
```

Here, we have used

```
JUnitCore
```

class of JUnit to run the test cases. This is required as we need to add our custom listener to the test cases. See the highlighted line in above class.

3.2.1. Output

```
01 Test cases to execute : 4
02 Execution Started : test_A_1
03 Execution Failure : java.lang.AssertionError
04 Execution Finished : test_A_1
05 Execution Started : test_A_2
06 Execution Finished : test_A_2
07 Execution Started : test_B_1
08 Execution Finished : test_B_1
09 Execution Ignored : test_B_2
10 Test cases executed : 3
```

It is cleared from the output that with each test case,

```
testStarted()
```

and

```
testFinished()
```

methods are called.
One of the test is failed due to the condition which we have passed in

```
TestClassA
```

class.

4. Conclusion

We have learnt in this example, that by using a custom listener in JUnit we can log and do tasks accordingly on the basis of methods executed. Like, if you want to call or notify the user that a particular test case is failed, you can simply write that piece of code in

```
testFailure()
```