TESTNG - GROUP TEST

Group test is a new innovative feature in TestNG, which doesn't exist in JUnit framework. It permits you to dispatch methods into proper portions and perform sophisticated groupings of test methods. Not only can you declare those methods that belong to groups, but you can also specify groups that contain other groups. Then, TestNG can be invoked and asked to include a certain set of groups *orregularexpressions*, while excluding another set. Group tests provide maximum flexibility in how you partition your tests and doesn't require you to recompile anything if you want to run two different sets of tests back to back.

Groups are specified in your testng.xml file using the <groups> tag. It can be found either under the <test> or <suite> tag. Groups specified in the <suite> tag apply to all the <test> tags underneath.

Now, let's take an example to see how group test works.

Create a Class

- Create a java class to be tested, say, MessageUtil.java in **C:\ > TestNG_WORKSPACE**.

```
/*
* This class prints the given message on console.
*/
public class MessageUtil {
   private String message;

   // Constructor
   // @param message to be printed
   public MessageUtil(String message) {
      this.message = message;
   }

   // prints the message
   public String printMessage() {
      System.out.println(message);
      return message;
   }

   // add "tutorialspoint" to the message
   public String salutationMessage() {
      message = "tutorialspoint" + message;
      System.out.println(message);
      return message;
   }

   // add "www." to the message
   public String exitMessage() {
      message = "www." + message;
      System.out.println(message);
      return message;
   }
}
```

Create Test Case Class

- Create a java test class, say, GroupTestExample.java.
- Add test methods, testPrintMessage and testSalutationMessage, to your test class.
- Group the test method in two categories:
    - Check-in tests *checkintest*: These tests should be run before you submit new code. They should typically be fast and just make sure no basic functionality is broken.

- Functional tests *functest*: These tests should cover all the functionalities of your software and be run at least once a day, although ideally you would want to run them continuously.

Create the java class file named GroupTestExample.java in **C:\>TestNG_WORKSPACE**.

```
import org.testng.Assert;
import org.testng.annotations.Test;

public class GroupTestExample {
   String message = ".com";
   MessageUtil messageUtil = new MessageUtil(message);

   @Test(groups = { "functest", "checkintest" })

   public void testPrintMessage() {
      System.out.println("Inside testPrintMessage()");
      message = ".com";
      Assert.assertEquals(message, messageUtil.printMessage());
   }

   @Test(groups = { "checkintest" })

   public void testSalutationMessage() {
      System.out.println("Inside testSalutationMessage()");
      message = "tutorialspoint" + ".com";
      Assert.assertEquals(message, messageUtil.salutationMessage());
   }

   @Test(groups = { "functest" })

   public void testingExitMessage() {
      System.out.println("Inside testExitMessage()");
      message = "www." + "tutorialspoint"+".com";
      Assert.assertEquals(message, messageUtil.exitMessage());
   }
}
```

Create testng.xml

Create testng.xml in **C:\ > TestNG_WORKSPACE**, to execute test case*s*. Here, we would be executing only those tests, that belong to the group *functest*.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd" >

<suite name="Suite1">
   <test name="test1">

   <groups>
      <run>
      <include name="functest" />
      </run>
   </groups>

   <classes>
      <class name="GroupTestExample" />
   </classes>

   </test>
</suite>
```

Compile the MessageUtil, Test case classes using javac.

```
C:\TestNG_WORKSPACE>javac MessageUtil.java GroupTestExample.java
```

Now, run the testng.xml, which will run only the method testPrintMessage, as it belongs to the

group *functest*.

```
C:\TestNG_WORKSPACE>java -cp "C:\TestNG_WORKSPACE" org.testng.TestNG testng.xml
```

Verify the output. Only the method testPrintMessage is executed.

```
Inside testPrintMessage()
.com
Inside testExitMessage()
www..com

===============================================
Suite1
Total tests run: 2, Failures: 1, Skips: 0
===============================================
```

Group of Groups

Groups can also include other groups. These groups are called *MetaGroups*. For example, you might want to define a group *all* that includes *checkintest* and *functest*. Let's modify our testng.xml file as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd" >
<suite name="Suite1">
   <test name="test1">

      <groups>

         <define name="all">
            <include name="functest"/>
            <include name="checkintest"/>
         </define>

         <run>
            <include name="all"/>
         </run>

      </groups>

      <classes>
         <class name="GroupTestExample" />
      </classes>

   </test>
</suite>
```

Executing the above testng.xml will execute all the three tests and will give you the following result:

```
Inside testPrintMessage()
.com
Inside testSalutationMessage()
tutorialspoint.com
Inside testExitMessage()
www.tutorialspoint.com

===============================================
Suite1
Total tests run: 3, Failures: 0, Skips: 0
===============================================
```

Exclusion Groups

You can ignore a group by using the <exclude> tag as shown below:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd" >
<suite name="Suite1">
   <test name="test1">

      <groups>
         <define name="all">
            <exclude name="functest"/>
            <include name="checkintest"/>
         </define>

         <run>
            <include name="all"/>
         </run>
      </groups>

      <classes>
         <class name="GroupTestExample" />
      </classes>

   </test>
</suite>
```