TESTNG - DEPENDENCY TEST

Sometimes, you may need to invoke methods in a test case in a particular order, or you may want to share some data and state between methods. This kind of dependency is supported by TestNG, as it supports the declaration of explicit dependencies between test methods.

TestNG allows you to specify dependencies either with:

- Using attribute *dependsOnMethods* in @Test annotations, OR.
- Using attribute *dependsOnGroups* in @Test annotations.

Example Using *dependsOnMethods*

Create a Class

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

```
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 "Hi!" to the message
   public String salutationMessage() {
      message = "Hi!" + message;
      System.out.println(message);
      return message;
   }
}
```

Create Test Case Class

- Create a java test class, say, DependencyTestUsingAnnotation.java.
- Create a java test class, say, DependencyTestUsingAnnotation.java.
- Add test methods, testPrintMessage and testSalutationMessage, and initEnvironmentTest, to your test class.
- Add attribute *dependsOnMethods = {"initEnvironmentTest"}* to the @Test annotation of *testSalutationMessage* method.

Create a java class file name DependencyTestUsingAnnotation.java in **C:\>TestNG_WORKSPACE**.

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

public class DependencyTestUsingAnnotation {
String message = "Manisha";
MessageUtil messageUtil = new MessageUtil(message);
```

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

   @Test(dependsOnMethods = { "initEnvironmentTest" })
   public void testSalutationMessage() {
      System.out.println("Inside testSalutationMessage()");
      message = "Hi!" + "Manisha";
      Assert.assertEquals(message, messageUtil.salutationMessage());
   }

   @Test
   public void initEnvironmentTest() {
      System.out.println("This is initEnvironmentTest");
   }
}
```

Create testng.xml

Create testng.xml in **C:\>TestNG_WORKSPACE** to execute test cases.

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

<suite name="Suite1">
   <test name="test1">
      <classes>
         <class name="DependencyTestUsingAnnotation" />
      </classes>
   </test>
</suite>
```

Compile the MessageUtil, Test case classes using `javac`.

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

Now, run the testng.xml, which will run the *testSalutationMessage* method only after the execution of *initEnvironmentTest* method.

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

Verify the output.

```
This is initEnvironmentTest
Inside testPrintMessage()
Manisha
Inside testSalutationMessage()
Hi!Manisha

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

Example Using *dependsOnGroups*

You can also have methods that depend on entire groups. Let's have an example to demonstrate this.

Create a Class

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

```
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 "Hi!" to the message
   public String salutationMessage() {
      message = "Hi!" + message;
      System.out.println(message);
      return message;
   }
}
```

Create Test Case Class

- Create a java test class, say, DependencyTestUsingAnnotation.java.
- Add test methods, testPrintMessage testSalutationMessage, and initEnvironmentTest to your test class, and add them to the group "init".
- Add the attribute *dependsOnMethods = {"init.*"}* to the @Test annotation of *testSalutationMessage* method.

Create a java class file named DependencyTestUsingAnnotation.java in **C:\>TestNG_WORKSPACE**.

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

public class DependencyTestUsingAnnotation {
   String message = "Manisha";
   MessageUtil messageUtil = new MessageUtil(message);

   @Test(groups = { "init" })
   public void testPrintMessage() {
      System.out.println("Inside testPrintMessage()");
      message = "Manisha";
      Assert.assertEquals(message, messageUtil.printMessage());
   }

   @Test(dependsOnGroups = { "init.*" })
   public void testSalutationMessage() {
      System.out.println("Inside testSalutationMessage()");
      message = "Hi!" + "Manisha";
      Assert.assertEquals(message, messageUtil.salutationMessage());
   }

   @Test(groups = { "init" })
   public void initEnvironmentTest() {
      System.out.println("This is initEnvironmentTest");
   }
}
```

In this example, testSalutationMessage is declared as depending on any group, matching the regular expression "init.*", which guarantees that the methods testPrintMessage and initEnvironmentTest will always be invoked before testSalutationMessage.

> *If a method depended upon fails, and you have a hard dependency on it* $always Run = false$, *whichisthedefault, the methods that depend on it are not marked as FAIL but as SKIP. Skipped methods will be reported as such in the final report* $inacolorthatisneitherRednorGreeninHTML$, *which is important since skipped methods are not necessarily failures.*

Create testng.xml

Create testng.xml in **C:\>TestNG_WORKSPACE** to execute test case*s*.

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

<suite name="Suite1">
   <test name="test1">
      <classes>
         <class name="DependencyTestUsingAnnotation" />
      </classes>
   </test>
</suite>
```

Compile the MessageUtil, Test case classes using `javac`.

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

Now, run the testng.xml, which will run the *testSalutationMessage* method only after the execution of *initEnvironmentTest* method.

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

Verify the output.

```
This is initEnvironmentTest
Inside testPrintMessage()
Manisha
Inside testSalutationMessage()
Hi!Manisha

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

dependsOnGroups Vs *dependsOnMethods*

- On using groups, we are no longer exposed to refactoring problems. As long as we don’t modify the dependsOnGroups or groups attributes, our tests will keep running with the proper dependencies set up.
- Whenever a new method needs to be added in the dependency graph, all we need to do is put it in the right group and make sure it depends on the correct group. We don’t need to modify any other method