Test NG

Class 3

Agenda

Test case grouping in TestNG

Including and excluding groups

Dependencies in TestNG

Test Case Grouping

- In TestNG users can group multiple test methods into a named group. We can also execute a particular set of test methods belonging to a group or multiple groups.

- This feature allows the test methods to be **segregated into different sections or modules**. For example, we can have a set of tests that belong to smoke test whereas others may belong to regression tests.

- We can also **segregate the tests based on the functionalities/features** that the test method verifies. This helps in executing only a particular set of tests as and when required.

Test Case Grouping

- '**Groups**' is one more annotation of TestNG which can be used in the execution of multiple tests. which doesn't exist in JUnit framework.

- Using TestNG we can execute only set of groups while excluding another set. **This gives us the maximum flexibility in divide tests and doesn't require us to recompile anything if you want to run two different sets of tests back to back.**

- Groups are specified in testng.xml file and can be used either under the or tag. Groups specified in the tag apply to all the tags underneath.

- TestNG groups allow grouping of test methods. In TestNG, a test method can belong to more than one group and likewise, a group can consist of multiple test methods.

- 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.

Test Case Grouping

```
import org.testng.annotations.Test;
public class TestGroupExample
{
	@Test(groups = { "test-group" })
	public void testMethodOne() {
		SOP("Test method one belonging to group.");
	}
	@Test
	public void testMethodTwo() {
		SOP("Test method two not belonging to group.");
	}
	@Test(groups = { "test-group" })
	public void testMethodThree() {
		SOP("Test method three belonging to group.");
	}
}
```

Note: If we will run above test in eclipse normally, then test execution does not consider the group for execution and hence executes all the tests in the specified test class.

How to specify group through testng.xml

```
<suite name="Time test Suite" verbose="1">
  <test name="Group Test">
    <groups>
      <run>
        <include name="test-group" />
      </run>
    </groups>
    <classes>
      <class name="packageName.TestGroupExample" />
    </classes>
  </test>
</suite>
```

This method is the preferred and easy way to execute groups.

Assigning multiple Groups

```
import org.testng.annotations.Test;
public class MultiGroupExample
{
	@Test(groups = { "group-one" })
	public void testMethodOne() {
		SOP("Test method one belonging to group.");
	}
	@Test(groups = { "group-one", "group-two" })
	public void testMethodTwo() {
		SOP("Test method two belonging to both group.");
	}
	@Test(groups = { "group-two" })
	public void testMethodThree() {
		SOP("Test method three belonging to group.");
	}
}
```

TestNG allows methods belong to multiple groups also. This can be done by providing the group names as an array in the groups attribute of the @Test annotation.

Including and excluding groups

- TestNG also allows to **include and exclude certain groups from test execution**. This helps in executing only a particular set of tests and excluding certain tests.

- A simple example can be when a feature is broken and you need to exclude a fixed set of tests from execution since these test will fail upon execution.

- Once the feature is fixed you can then verify the feature by just executing the respective group of tests.

- We can specify the include and exclude in testng.xml.
  - **<include> –** It tells testng.xml that which group we need to execute.
  - **<exclude>-** It tells testng.xml which group we have to Skip

Including and excluding groups

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd">
<suite name="My First Suite">
    <test name="My First Test">
        <groups>
            <run>
                <include name="Group 1" />
                <exclude name="Group 3" />
            </run>
        </groups>
        <classes>
            <class name="com.Class2.MultiGroupExample"/>
        </classes>
    </test>
</suite>
```

Task

Identify Priority of Test Cases

TC 1: Saucedemo Username1(tag the groups - Smoke1, Regression1)

Step 1: Open browser and navigate to Saucedemo

Step 2: Enter username standard_user and enter password secret_sauce and click login button

Step 3: Verify user successfully logged in

TC 2: Saucedemo Username2(tag the groups - Smoke2, Regression2)

Step 1: Open browser and navigate to Saucedemo

Step 2: Enter username problem_user and enter password secret_sauce and click login button

Step 3: Verify user successfully logged in

Note: Create BeforeMethod and AfterMethod annotations to open browser and logout from the application. Create a xml file and include smoke1, regression2 and exclude smoke2, regression1.

Test Dependency in TestNG

Sometime we need to run test methods in certain order, as we need to run a test method that always runs after some other test method.

This kind of dependency is supported by TestNG and can be implemented using:

- **dependsOnMethods in @Testannotations.**
- **dependsOnGroups in @Testannotations.**

Test Dependency in TestNG

```
public class DependencyTest {
@Test
public void start() {
SOP("Starting the server");
}

@Test(dependsOnMethods = { "start" })  // Set dependency on
start method
public void init() {
SOP("Initializing the data for processing!");
}

@Test(dependsOnMethods = { "start", "init" }) // Set Dependencies
on start and init tests
public void process() {
SOP("Processing the data!");
}

@Test(dependsOnMethods = { "process" })
public void stop() {
SOP("Stopping the server");
}
```

Task

Identify Priority of Test Cases

TC 1: WebOrder Verify Successful Login ( )
Step 1: Open browser and navigate to WebOrder
Step 2: Enter valid username, enter valid password and click on the login button
Step 3: Verify user successfully logged in

TC 2: WebOrder Creating and verifying Users( )
Step 1: Create Two users and fill out all the required fields
Step 2: Verify that user one name appears within the table
Step 3: Edit user one and update user one's State, assert the new updated State is displayed and save the changes.
Step 4: Verify that user two name appears within the table
Step 5: Edit user two and update user two's City and assert the new updated City is displayed and save the changes.

TC 3: WebOrder verifying Users( )
Step 1 : Assert Both User one and User Two are displayed

Note: Create BeforeClass and AfterClass annotations to open browser and logout from the application. The creation of users two should depend on the creation of users one.The validation both users should depend on the creation of both users. Create xml file and share a screenshot of the test.