TEST AUTOMATION UNIVERSITY

Rex Jones II

Test Next Generation (TestNG)
A Powerful Test Framework

Chapter 0: Course Overview

Hi, my name is Rex Jones II and Welcome To Test Automation University. In this course, I will cover Test Next Generation (TestNG) A Java Test Framework. You will see why TestNG was designed for developers and automation testers who want to go beyond unit testing. xUnit is a family of Unit Test Frameworks that consist of JUnit for Java, PyUnit for Python, NUnit for the .Net programming languages such as C Sharp.

JUnit is the most popular test framework but TestNG is the most powerful test framework. TestNG was influenced by JUnit so it adopted the same concepts then added more testing features.

Our lesson plan will include how to install TestNG. The steps are demonstrated using Eclipse but IntelliJ is also compatible with TestNG.

How to implement TestNG Annotations. Annotations have a special meaning for methods in a class.

After Annotations, we will take a look at TestNG Assertions. The purpose of Assertions is to verify whether our Test Passed or Failed.

Next is TestNG Attributes from the Test Class. There are many Attributes but we are going to look at the priority Attribute, dependsOnMethods Attribute, and dependsOnGroups Attribute. Both dependsOn Attributes help us to carry out many kinds of test types. Some of those test types are Dependency, Integration, Smoke, and Regression which separate TestNG from all other test frameworks.

In addition, Data Driven Testing and Cross Browser Testing are 2 more Test Types that makes TestNG very powerful. Data Driven Testing is when we store our Test Data then use that Test Data as input for our Test Script. Cross Browser Testing is the process of executing 1 Test Script on multiple browsers at the same time.

Now, let's Get Started With Test Next Generation (TestNG) A Java Test Framework.

Chapter 1: Introduction To TestNG

Introduction

Hello and Welcome To Chapter 1, Introduction To TestNG. In this chapter, we will discuss What Is A Test Framework, Why Test Frameworks Are Important, the xUnit Family of Unit Test Frameworks, and What Separates TestNG From Other Test Frameworks.

What Is A Test Framework

What Is A Test Framework? A Test Framework is a pattern for writing Test Scripts and running Test Scripts. The purpose is to monitor testing and development of an (AUT) Application Under Test.

A Test Framework is different from an Automation Design Framework. Some of the most popular Automation Design Frameworks are Data Driven Frameworks, Keyword Driven Frameworks, and Hybrid Driven Frameworks. With Selenium, we utilize a Test Framework to create our Automation Design Framework.

Why Are Test Frameworks Important

Why Are Test Frameworks Important? Test Frameworks are important because it simplifies the process of testing. How? How does it simplify the process? One of the ways, Test Frameworks simplify the process is by giving power to developers and automation engineers to write a quick test. We can add an annotation and assert statement then start our test. Another way Test Frameworks simply the process is how we run our test. We have the ability to run our test as a collection in a Test Suite.

The xUnit Family of Frameworks

The xUnit Family of Test Frameworks. xUnit is a name for a compilation of Unit Testing Frameworks. Some of those frameworks are JUnit, PyUnit, and NUnit. JUnit is for Java, PyUnit is for Python, and NUnit is for the .Net Programming languages such as C#. The xUnit family became popular after releasing JUnit. Later, many programming languages embraced the common architecture and joined the xUnit family.

Core Functions of A Test Framework

Some of the core functions within a Test Framework include

- Creating and Executing Test Scripts
- Generating Test Reports
- Generating Logs
- Last but not least Reading and Writing Test Data. We can use Microsoft Excel or any other Test Artifact for reading and writing Test Data.

However, as their family name implies, xUnit is a Unit Testing Framework. A Unit Testing Framework can test units of an application at a time which is the smallest part of an application.

What Separates TestNG From Other Frameworks

What Separates TestNG From Other Frameworks? The ability to test more than a unit separates TestNG from all other Test Frameworks. TestNG has the same core functions

- Create and Execute Test Scripts
- Generate Test Reports
- Generate Logs
- Read & Write Test Data

just like the xUnit Family. TestNG can perform the same functions because it was influenced by JUnit 1, JUnit 2, and JUnit 3. In return, JUnit was also influenced by TestNG. JUnit 4 was released after TestNG and picked up a lot of the same concepts from TestNG.

The xUnit Family is good but TestNG have more testing features to perform Parallel Testing, Integration Testing, Dependency Testing, Data Driven Testing, Cross Browser Testing, End To End Testing, and more types of testing. TestNG is a step ahead of the rest when it comes to testing.

Next, in Chapter 2, we will install TestNG.

Chapter 2: How To Install TestNG

Introduction

Hello and Welcome To Chapter 2, How To Install TestNG. In this chapter, we will discuss the Various Ways To Install TestNG then show How To Install TestNG Using Eclipse Marketplace and How To Install TestNG Using an option called Install New Software in Eclipse.

Various Ways To Install TestNG

Let's start with the compatible IDE's. IDE stands for Integrated Development Environment. The IDE's are Eclipse, NetBeans, and IntelliJ. Next, we can setup TestNG using Build Tools Ant or Maven.

In addition to the IDE's and Build Tools, we can use the Command Line or download the TestNG JARs. The focus will be installing TestNG using Eclipse but I will provide a reference link for NetBeans, IntelliJ, Ant, and Maven.

How To Install TestNG Using Eclipse

Eclipse Marketplace

How To Install TestNG Using Eclipse? First, I'll show you to how install TestNG using Eclipse Marketplace. Go to Help, select Eclipse Marketplace, type TestNG in the Find text box, then click the Go button. We see TestNG for Eclipse has an Install button. Click the Install button then click the Confirm button after making sure all of the checkboxes are checked. Select I accept the terms of the license agreement and click Finish.

Next, a Security Warning shows up to let you know, you are installing software that contains unsigned content. This shows up because TestNG is 3rd Party plugin. Click Install anyway. The last step is restart Eclipse so we click Restart Now.

Demo – Install New Software

Another way to install TestNG is through Install New Software. We go to Help, select Install New Software then type TestNG, and click Enter.

If TestNG Eclipse does not come up for you as an auto-suggestion then click the Add button. In the Add Repository pop-up, enter TestNG for Name and http://beust.com/eclipse for Location, then click OK. Select the checkbox for TestNG. I already have TestNG installed so my Next button is disabled. You can verify if TestNG is installed by clicking the already installed link. We see a list of what's installed in Eclipse. TestNG

The Remaining Steps is to click the Next button, accept the License Agreement, then the click Finish button. Clicking the Finish button is the last step for installing TestNG. After installing TestNG, we must add the TestNG Library.

Add TestNG Library

We have the option of adding our TestNG Library now or waiting until we import an annotation. Let's go ahead and add the TestNG Library now. There's more than 1 way to add a TestNG Library but we are

going to Configure the Build Path. We can right click src, right click JRE System Library, or right click Referenced Libraries, select Build Path, Configure Build Path.

Select Classpath, click the Add Library button, select TestNG, click Next, click Finish and we see TestNG. Click the Apply and Close button. TestNG is located under the Test Automation University project. We are finished installing TestNG and adding TestNG as a Library.

Reference Links

Here's 5 Reference Links for installing and setting up TestNG. We have IntellJ which shows go to Preferences -> Plugins then click over to the Available Plugins tab. The other IDE is NetBeans. Follow the Installation section to setup TestNG. Next, we have Maven. Specify your pom.xml and your dependency should look like this. Then we have Ant. You must define the TestNG task. Last, we have a Reference Document for Eclipse. It has steps that can be downloaded which covers the same installation steps from the video.

Now, that we have installed TestNG. In Chapter 3, we will cover TestNG Annotations which manage how methods are run.

Chapter 3: TestNG Annotations

Chapter 3.1: TestNG Annotations Overview

Introduction

Hello and Welcome To Chapter 3, TestNG Annotations. This chapter is a series of 3 videos: An Overview of TestNG Annotations, Introduction To TestNG Annotations, TestNG Execution Flow, and we will view the TestNG xml File.

Chapter 3 Overview

This video is 3.1 an Overview of TestNG Annotations. Video 3.2 is an Introduction To TestNG Annotations. In that video, we will talk about What Are TestNG Annotations, How To Add TestNG Annotations, and the Configuration Annotations. Video 3.3 is a video about the Execution Flow of the TestNG Annotations and the TestNG xml File. Our execution flow depends on the annotations and the xml file helps understand the TestNG Suite, Test, Class, and Methods.

Next is video 3.2, an Introduction To TestNG Annotations.

Chapter 3.2: Introduction To TestNG

Introduction

Welcome To Chapter 3.2, Introduction To TestNG Annotations. In this chapter, we will discuss What Are TestNG Annotations, How To Add TestNG Annotations, and the TestNG Configuration Annotations.

What Are TestNG Annotations

What Are TestNG Annotations? A TestNG Annotation is data that have a special meaning for a Java method. It provides information about how the annotation will control the execution order. I see TestNG Annotations as Pre-Conditions, Conditions, and Post-Conditions. Why? Because our Automation

Test Scripts are very similar to Manual Test Cases. Sometimes with Manual Test Cases, there are Pre-Conditions that must be set up before we start our test. Our test is the Condition and after our test is the Post-Condition.

It's the same with automation, we have Pre-Conditions that must be set up before we test then we have our test. Our test is the Condition. Next is the Post-Condition that is performed after we complete our testing. Notice the @ symbol. In some places, you may hear @ symbol or @ sign. Both names are short for at a rate which is an accounting term. This symbol is placed in front of each TestNG Annotation. All of the Pre-Conditions begin with @Before while the Condition is @Test. @Test is a key annotation because it performs our test. Last, we have the Post-Conditions that all start with @After.

How To Add TestNG Annotations

How To Add TestNG Annotations? Let's go to Eclipse. There is more than 1 way to add an annotation. I believe most people start from scratch and write our annotation. For example, write @BeforeMethod then write our method. public void setup () and import the annotation. A short cut for importing and organizing our imports is CTRL + SHIFT + O. If you want to see a list of Eclipse shortcuts select CTRL + SHIFT + L and we see a lot of shortcuts including Organize Imports.

Another way to add TestNG Annotations is by selecting the package, right click, New then Other. Type TestNG, Select TestNG class then click Next. Do you see the Annotations? Let's select all of the annotations except for DataProvider. We are going to cover DataProvider in a subsequent chapter called Data Driven Testing. Data Driven Testing is when we run 1 Test Script with many sets of data. Change the Class Name to Configuration_Annotations and Click Finish.

Configuration Annotations

A Configuration Annotation is an annotation that begins with Before or After. They are called Configuration Annotations because the Before Annotations help us set up variables and configurations before starting test execution. The After Annotations help us clean up everything after executing our test. If we take a look at the TestNG Annotations package, we will see all of the annotations. The Configuration Annotations start at AfterClass and stop at BeforeTest.

In our Editor, the annotations are displayed in order from lowest to highest. Let's start with the highest level and add some print statements. BeforeSuite/AfterSuite run before a suite start and after all Test Methods. So, let's write BeforeSuite, Chrome - Set Up System Property. AfterSuite, Chrome – Clean Up All Cookies.

BeforeTest/AfterTest run before a test start and after all Test Methods. Now that we have set up Chrome, BeforeTest will Open Chrome and AfterTest will Close Chrome. BeforeClass/AfterClass run before a test class start and after all Test Methods. After opening Chrome, BeforeClass will Open Test Application and AfterClass will Close Test Application.

Next, we have BeforeMethod/AfterMethod which run before a Test Method and after a Test Method. Now that we finally opened the Test Application, let's Sign In and Sign Out. Our test will Search For Customer so change the name to searchCustomer. We can add as many test annotations as we want but let's add 1 more to Search For Product and change the name to searchProduct.

The annotations can be placed in any order on the editor because TestNG identifies the methods by looking up the annotation. For example, we can place BeforeSuite anywhere on this editor and it will always execute first.

Next in Chapter 3.3, we will see the order of how these annotations execute.

Chapter 3.3: TestNG Annotations Execution Flow & xml File

Introduction

Welcome To Chapter 3.3, TestNG Annotations Execution Flow and the xml File. In this chapter, we will discuss the Execution Flow of TestNG Annotations, View the TestNG xml File For TestNG Suite, Test, Class, & Method tags, then decide Which TestNG Annotations We Choose For Testing.

The TestNG Annotations dictate the execution order and the xml file is a great way to see why annotations execute in that order. We will take a look at some of the annotations to see why Test Requirements determine our TestNG Annotations.

TestNG Annotations Execution Flow

The execution flow depends on our annotations. Therefore, the methods execute according to the rank of each annotation. Let's go to Eclipse and change the annotation order.

We will change the order on the Editor to match the order in how they will show up on the Console. BeforeSuite, BeforeTest, BeforeClass, BeforeMethod, first Test Method Search Customer, second Test Method Search Product, AfterMethod, AfterClass, AfterTest, and AfterSuite. Let's Run.

We see BeforeSuite / Chrome – Set Up System Property, then we see BeforeTest / Open Chrome, next is BeforeClass / Open Test Application, and we have BeforeMethod / which is Sign In. Here's the interesting part, the BeforeMethod always run before the Test Method. In this case, the first Test Method is Search For Customer which has an @Test Annotation and the AfterMethod always run after the Test Method. The AfterMethod is Sign Out.

We finished searching for a customer. However, Chrome is still set up, Chrome is still open, and the application remains open but we are not signed into the application. Now we start over with the BeforeMethod, which is a Pre-Condition to Sign In before Searching For A Product then the Post-Condition, AfterMethod Sign Out of the application. This shows us how TestNG executes the Pre-Condition, the Condition, and the Post-Condition based on our annotations.

Next, we have AfterClass / Close Test Application, AfterTest / Close Chrome, and AfterSuite – Clean Up All Cookies. In the Console, we see both searchCustomer and searchProduct PASSED. The TestNG results tab also shows both tests PASSED.

View xml File To See TestNG Suite, Test, Class, & Methods

1 Test

View xml file. The purpose of an xml file is to store data and to carry data for all of our testing. We see a tag for suite, a tag for test, a tag for class, and a tag for methods. A Suite can have 1 or more tests. A Test can have 1 or more classes and a Class can have 1 or more methods. The xml file gives us a picture on why the annotations execute in a particular order.

Let's Run our Test Suite. We see the same output. It's the same output because this execution has 1 test tag. We see BeforeSuite, BeforeTest, BeforeClass, BeforeMethod executes before both Test Methods – Search For Customer and Search For Product, then the AfterMethod executes after both Test Methods.

Next is AfterClass, AfterTest, and AfterSuite. Do you see how the execution order is consistent with the xml file? Let's look at an xml file that has 2 test tags.

2 Tests

This xml file does not include the Test Methods together. We see the 1st Test Name is Search For A Customer and the 2nd Test Name is Search For A Product. Let's run. The output is different. This time our execution does not immediately Sign back into the application after Signing out of the application. Now, the execution processes AfterClass – Close Test Application and AfterTest – Close Chrome after signing out. Why? Because in the xml file, the Test Methods are not in included the same class. Therefore, execution will not sign back in to start our next test. Notice, Chrome is still set up after executing BeforeSuite. That means the AfterSuite annotation is the only annotation that have not been executed. That completes searching for a customer.

Next is Search For A Product which starts with BeforeTest – Open Chrome, BeforeClass – Open Test Application, then BeforeMethod – Sign In and Search For A Product. Finally, AfterMethod – Signs Out, AfterClass – Close Test Application, AfterTest – Close Chrome, now AfterSuite – Clean Up All Cookies from Chrome.

In reality, we probably would not use 4 Before Configuration Annotations and 4 After Configuration Annotations. However, I wanted you to see how the xml file is connected to the annotations and how the xml file allows us to execute the same methods in 1 test or more than 1 test.

Which Annotation(s) Do We Choose For Testing

Which annotations do we choose for testing? The annotations we choose for testing depends on our Test Requirements. Do we need our test to set up a Pre-Condition before every Test Method and clean up with a Post-Condition after every Test Method or do we need 1 Pre-Condition before all of the Test Methods and 1 Post-Condition after all of the Test Methods? Let's walkthrough our Test Application then I will show you the difference using 2 pairs of Configuration Annotations and 3 Test Methods.

The Test Requirement is to sign in with Username Admin and Password admin123 then click the Login button. After signing in, we click the Admin tab then search for a user. Finally, we sign out.

Our code uses a BeforeMethod annotation and AfterMethod annotation. First, we setup the test, then sign into the application, after signing into the application, we search for a user and sign out. After signing out we teardown our test which is close the browser.

Let's run and see what happens. We see 2 failures in the Console. The results tab also shows 2 Failures. Go back to the Console and look at our first Test signIn. Step 1 shows we opened Chrome and the application, Step 2 shows we signed into the application, but Steps 3 and 4 are missing before Step 5 close Chrome and the application. It passed but did not perform the steps we expected.

Let's look at the next test userSearch. Step 1 Open Chrome & the application. Steps 2, 3, and 4 are missing but we see Step 5 which Close Chrome and the application. Why did the Test Script skip steps 2 through 4? It skipped those steps because we cannot open Chrome and the application then search for a user. We cannot search for a user because we have not signed into the application. The console shows Failed: userSearch. Unable to locate element. It could not locate the Admin tab to begin searching for a user.

Same with the userSignOut. Our Test Script could not sign out of the application because it never signed into the application. Steps 2, 3, and 4 are missing. If we scroll down, the console shows FAILED: userSignOut. With this Test Requirement, we must use BeforeClass/AfterClass or BeforeTest/AfterTest. So, our code does not open the application before every Test Method and close the application after every Test Method.

Now watch what happens when we use the same code but change the Configuration Annotations to BeforeClass and AfterClass. Let's Run. All 3 Test Scripts PASSED. The results tab shows all 3 PASSED. The Console shows Step 1, we opened Chrome and the application, Step 2, we signed into the application, Step 3, we searched for a user, Step 4 we signed out of the application, and Step 5, we closed Chrome and the application.

TestNG provides a lot of annotations for our Test Requirements. This is a list of the annotations we covered in Chapter 3. Next in Chapter 4, we will see how the Test Methods execute using a priority attribute.

Chapter 4: Priority Attribute

Introduction

Hello and Welcome To Chapter 4, Priority Attribute. In this chapter, we will discuss the Default Execution Order For Test Methods and the Priority Attribute For The Test Methods.

Default Execution Order For Test Methods

The Default Execution Order For Test Methods. A Test Method is marked by the @Test Annotation. We can set our methods as a Test Method and we can set our class as a test.

Class Marked By Test Annotation

Class Marked By Test Annotation. Let's go to Eclipse. In this example, I marked the class as part of the TestNG Test by adding a Test annotation at the class level. setup and teardown have Configuration Annotations that uses the BeforeClass and AfterClass annotations. signIn does not have an annotation. Also, searchTShirt and signOut do not have an annotation. Looking at this layout, it seems like the execution order should be setup, signIn, searchTShirt, signOut, then teardown. However, that's not the case, execution is going to run searchTShirt before signIn and signOut.

Why will it execute searchTShirt before signIn and signOut? It will execute searchTShirt first because the order depends on the names of each method. TestNG runs the program in ascending alphabetical order from A – Z. Therefore, in ascending order, se in searchTShirt comes before si in signIn and signOut. We

can place the Test Methods anywhere on the editor and it will run in the same order every time. It will run in the same order because the Test annotations identify the Test Methods.

Before running this program, let's walkthrough the Test Application. The plan is to first Sign In, by entering an Email of TestNG@Framework.com and Password of TestNG1234 then click the Sign button. After clicking the Sign In button, we click T-Shirts, search for a Blue T-shirt, then click the Search button. Last, we Sign Out.

Now, let's run our Test Script. We see the order shows Number 2 Search For T-Shirt, Number 1 Sign In, and Number 3 Sign Out. The same order shows up in the Results tab. As a side note, only the methods with a public access modifier are marked as Test Methods when marking the class with a Test Annotation. Any other access modifier will not set the method as a Test Method. If I change one of these methods to private then that method will not show up in the Console or Results tab. Know what, I'm going to change 2 methods. Sign Out will have a default access modifier. Let's run. Only Step 2 Search For T-Shirt shows up. Change both methods back to public.

Methods Marked By Test Annotation

Methods Marked By Test Annotation. It's the same when adding a Test annotation at the method level. In this example, the class is not marked as a test but all methods except for setUp and teardown are marked as Test Methods. Sign In has a Test annotation, Search T Shirt has a Test annotation, and Sign Out has a Test annotation. I'm going to run and we will see the same execution order. 2, 1, 3: Search For T-Shirt, Sign In, and Sign Out

Priority Attributes For Test Methods

Priority Attribute For Test Methods. The purpose of a priority attribute is to determine the execution order for our Test Method. The Test annotation has a lot of attributes. We can see those attributes by going to the TestNG Library, TestNG jar file, org.testng, annotations, scroll down to the Test.class then we see the attributes. Priority uses an integer Data Type.

We use a priority attribute by writing priority within a parenthesis after the Test annotation. The lowest number gets executed first. Some people start at zero but I prefer to use 1 since it's the first Test Method. The next Test Method is Search T Shirt which will be priority equal 2. The last Test Method is Sign Out and that will have a priority of 3. Let's run.

Now, we see the correct order. Step 1 Sign, Step 2 Search For T Shirt and Step 3 Sign Out. The Priority attribute helps us order our Test Methods. Next in Chapter 5, we will cover TestNG Assertions which verify if our test Pass or Fail.

Chapter 5: TestNG Assertions

Chapter 5.1 – Introduction To TestNG Assertions

Introduction

Hello and Welcome To Chapter 5, TestNG Assertions. This chapter is a series of 3 videos. The videos are Chapter 5.1 - Introduction To TestNG Assertions, Chapter 5.2 - Hard Asserts, and Chapter 5.3 - Soft Asserts.

In this Chapter 5.1, Introduction To TestNG Assertions. We will discuss What Are TestNG Assertions, View The TestNG Assertion Methods, and see why JUnit Assertions and TestNG Assertions are similar to each other. Before discussing assertions, let's go to Eclipse and see why we need assertions.

We will start by running the same Test Script from Chapter 3.3 which setup our test by opening Chrome and the OrangeHRM application, sign in, search for a user, sign out, then tear down our test by closing Chrome and the application. Let's Run.

All 3 Tests Passed but I have a question for you. Up to this point, what have we tested. We have automated Opening Chrome and the application, signing into the application, searching for a user, signing out, closing chrome, and closing the application. We have not verified our Test Script and not sure if it truly Passed or Failed. The Console shows Passed and the Results tab shows all 3 Test Passed. However, it shows Passed because there was no error in our automation code and no error when running our Automation Test Script. Let me show you something else. I'm going to remove all code to search for a user then Run. The Test Script did not search for a user but look what the Console shows PASSED: userSearch although Number 3 Search For User is missing. The Results tab also shows Passed for userSearch. That's not right.

What Are TestNG Assertions

What Are TestNG Assertions? TestNG Assertions verify if our test truly Passed or Failed. It's a line of code that is placed in our Test Method to verify a condition.

TestNG Assertion Methods

TestNG Assertion Methods. There are many assertions for TestNG but most of them are overloaded versions of the following methods: assertTrue, assertFalse, assertSame, assertNotSame, assertNotNull, and assertEquals. Generally, all of these TestNG Assertions have the same 3 parameters: Actual Result, Expected Result, and a String. It's the same with JUnit Assertions which have an assertion class located in TestNG's distribution.

JUnit & TestNG Assertions

JUnit has a class called junit.framework.Assert that have similar overloaded methods. TestNG turned around and added the same JUnit class called org.testng.AssertJUnit to its distribution. Why did TestNG add the same class as JUnit? TestNG added the same class as JUnit to guarantee all assertions keep working if we migrate our test from JUnit to TestNG. TestNG also added another class called org.testng.Assert.

The main difference between JUnit's class and TestNG's class is the syntax. Their parameters are available in reverse order. For example, the assertEquals method for JUnit has a String as the first

parameter followed by an expected result then an actual result. The same method for TestNG has actual as the first parameter, expected as the second parameter then String as the last parameter.

Let's go to Eclipse and I'll show you the assertions. Go to the TestNG Library, maximize the TestNG jar file, select org.testng package, and we see both classes: Assert and AssertJUnit class. Maximize Assert and there's a lot of methods. Maximize AssertJUnit and we some of the same overloaded methods. The methods within our Assert class are considered Hard Asserts. Next in Chapter 5.2, we will cover Hard Asserts.

Chapter 5.2 - Hard Asserts

Hard Asserts

Welcome To Chapter 5.2, Hard Asserts. In the previous Chapter 5.1, we ran our Test Script and it showed Passed without a verification step. The Test Script showed Pass although there is no code in this userSearch Test Method. I'm going to add the code back.

Let's pretend our Test Method requires us to verify the Home Page for OrangeHRM. Go to OrangeHRM and walkthrough the steps. Sign In – Admin / admin123 then click the Login Button. Here's the Home Page. We are going to verify the Welcome hyperlink, Admin tab, and Dashboard. Go back to Eclipse and change the step numbers for userSearch, userSignOut, and teardown. Search User will be 6, User Sign Out will be 7, and Number 8 will be teardown Close Chrome and Application.

Let's add a new Test Method between signIn and userSearch then add some assertions. @Test public void testHomePageVerification. Assert is located in the TestNG class so we write Assert dot and we see the same assertions from Chapter 5.1. Our assertion will be assertEquals.

There're so many methods but they are overloaded. Select an assertion with a String message. We see actual, expected, and message. Actual is true and Expected is true. This assertion will pass because both parameters are equal. The message only shows up if there is a failure so let's write the message like there is an assertion failure. The Welcome Link Is Not Correct On The Home Page. Print statement 3 Verify Welcome Link.

Verify Admin tab. Assert.assertFalse. The description shows Asserts that a condition is false. If it is not an an AssertionError, with the given message, is thrown. Pass this assertion by writing false with a message that states The Admin Tab Is Not Displayed On The Home Page. In order for this assertion to pass the condition must return false. sysout Verify Admin Tab

Verify Dashboard. Assert.assertTrue. Pass this assertion by writing true with a message that states The Dashboard Is Not Correct On The Home Page. sysout 5 Verify Dashboard. This test will Pass. Let's Run. We see all 4 Test Methods Passed. All of the Steps 1 – 8 are printed on the Console including Number 3 Verify Welcome Link, Number 4 Verify Admin Tab, and Number 5 Verify Dashboard. The Results tab also shows each Test Method Passed.

Now, let's see what happens when this Test Method Fails. I'm going to Fail assertEquals for the Welcome Link and assertTrue for the Dashboard. Make the expected result false for assertEquals and the condition false for assertTrue then Run. As expected, testHomePageVerification FAILED.

Let's look at the steps. Steps 3, 4, and 5 are missing. The AssertionError shows our message because our Test failed. The Welcome Link Is Not Correct On The Home Page expected false but found true. Scroll to the bottom and we see no more AssertionErrors. Why is there only 1 AssertionError when 2 assertions failed? The Results tab also shows a Failure with 1 AssertionError.

There is only 1 AssertionError because we used a Hard Assert. A Hard Assert stops immediately after a Failure then move on to the next annotation. In this case, assertEquals statement failed and skipped the subsequent assertions: assertFalse and assertTrue then moved on to the Test Method userSearch which has an @Test annotation. That's not good and that's why we did not see a Print Statement for Steps 3, 4, and 5. For this reason, TestNG introduced Soft Asserts.

Soft Alerts are located in the asserts package and there's only 2 methods: assertAll and doAssert. However, the Soft Assert class can extend the Assertion class so it can use all of these methods.

Next in Chapter 5.3, we will cover Soft Asserts.

Chapter 5.3 - Soft Asserts

Introduction

Welcome To Chapter 5.3, Soft Asserts. In this chapter, we will discuss the Difference Between Hard Asserts & Soft Asserts and answer the question Should Automation Engineers Use Hard and/or Soft Asserts.

Difference Between Hard Asserts & Soft Asserts

The difference between Hard Asserts and Soft Asserts is a Hard Assert stops executions after a fail and move to the next annotation. It does not matter if the next annotation is a Test Annotation or a Configuration Annotation.

We saw in the previous chapter 5.2 steps 3, 4, and 5 did not execute because they were inside the same annotation that failed. Therefore, those steps were skipped and execution picked up at the next Test Method which starts at step 6.

Someone might consider using a try-catch block for the Hard Assert but it won't work either. I'm going to show you. Here's the try-catch block which will try the assertions and catch the AssertionError. Let's Run. Look all of the Test Methods Passed although the AssertionError states The Welcome Link Is Not Correct On The Home Page expected false but found true. Scroll the console. Steps 3, 4, and 5 were skipped again. The Results tab shows all Test Methods Passed. That's not accurate.

A Soft Assert is different from a Hard Assert. It continues execution after a failed assertion and moves to the next statement line. It was designed to keep executing even when a verification step fails.

To use Soft Assert, first, we declare SoftAssert which is a class in TestNG then our object reference softassert equals new SoftAssert. Next, we replace all of the Asserts with our object reference softassert. The assertion methods (assertEquals, assertFalse, and assertTrue) will stay the same.

Recall from the previous video, that the softassert class had only 2 methods: assertAll and doAssert. We need to use assertAll every time for softassert to work. If assertAll is not used then our test will pass and

throw no AssertionError. I'm going to run without assertAll then run with assertAll to show you the difference. Run.

All Test Methods Passed and we know Steps 3 and 5 did not Pass. SoftAssert kept executing although the verification step Failed. Now, let's add assertAll to the end of the Test Method: softassert.assertAll and Run. One of our Test Methods Failed which is testHomePageVerification. Do you see both Assertions? The Welcome Link Is Not Correct On The Home Page expected false but found true. The Dashboard Is Not Correct On The Home Page expected true but found false. All 8 steps show up. The Results tab shows 1 Test Method Failed with the same 2 messages. The Welcome Link Is Not Correct On The Home Page expected false but found true. The Dashboard Is Not Correct On The Home Page expected true but found false. The Welcome Link and Dashboard are not correct on the Home page.

Here's a diagram that shows assertAll. On the left, we see a few assertion methods and assertAll on the right. The arrows are pointing to assertAll because they indicate an AssertionError gets stored into assertAll if there is a failure. That's why we place the assertAll method at the end of our Test Method.

Hard Asserts vs Soft Asserts

Hard Asserts versus Soft Asserts. Which one should we use for automation? We should use both asserts but it depends on our test. Sometimes it's good to have a hard assertion and other times it's not good to have a hard assertion. The same with soft assertions. Let's consider these print statements.

Open Browser, Open Application, Sign Into The Application, Go To Home Page and Verify Home Page, Go To Search Page and Verify Search Page, Search For User, and Sign Out of the Application. We know Hard Asserts stop if a verification step fails and will not execute the next statement but Soft Assert will execute the next statement.

If we had a failure after opening the browser then we would not want to continue executing the next step. There is no reason. With that scenario, it's best to implement a Hard Assert. The same with opening an application. There is no reason to keep executing our Test Script if it's a failure opening the application. The next step which is Sign Into The Application would return a failure.

However, we would not implement a Hard Assert after Step 4, Go To Home Page and Verify Home Page. If there's a failure on the Home Page, we still want to verify Step 5, Go To Search Page and Verify Search Page. To sum up assertions, we want to implement Hard Asserts in our code where it does not make sense to keep executing our test after a verification failure. We implement Soft Asserts in our code when there is a failure and we want to continue executing our test. Next in Chapter 6, we will cover Dependency Testing.

Chapter 6: Dependency Testing

Chapter 6.1 – Introduction To Dependency Testing

Introduction

Hello and Welcome To Chapter 6, Dependency Testing. Dependency Testing is one of the main features that separate TestNG from all other Test Frameworks. After reading Next Generation Java Testing, I

came across a section called Dependent Code and it mentioned some developers in the testing community are strongly opposed to any hint of dependencies in their test.

One of their arguments state as soon as Test Methods depend on other Test Methods, it becomes hard to run those methods in isolation. That argument was true until TestNG. Now, we are allowed to run Test Methods in isolation and part of a dependency. TestNG provides a way to clearly indicate those dependencies. Therefore, it has no problem considering the requirements for any Test Method we want to run in isolation.

This chapter is a series of 3 videos. The first video is Chapter 6.1, Introduction To Dependency Testing. The second video is Chapter 6.2 and it will cover the Depends On Methods Attribute. The third video is Chapter 6.3 which will cover the Groups and Depends On Groups Attribute.

Attributes are used to assist annotations with our testing. All 3 of these attributes can be placed in the Configuration Annotations and the Test Annotation. This chapter is 6.1 and we are going to cover Introduction To Dependency Testing.

What Is Dependency Testing

What Is Dependency Testing? Dependency Testing is when 2 or more actions, 2 or more tasks, 2 or more functions in our Test Application depend on the order of their methods. For example, we must Sign In to the application before we can Sign Out of the application. Therefore, signing out depends on us first signing in. If we, reverse that order and execute, then the sign out method would fail.

We have seen a form of Dependency Testing in Chapter 3 with Annotations. The Test Methods depend on the Configuration Annotations. If a Configuration Annotation fail then the Test Methods would not pass. Here's an example of a Failed Configuration. Let's run.

We see 1 Configuration Failure and 1 Skip. The BeforeClass setUp Configuration Failed and the AfterClass tearDown Configuration was Skipped. Then we see the Test Methods. There are 3 Runs and 3 Skips. Zero Failures. The Test Methods were skipped because the setUp method Failed. At the core, that is Dependency Testing. It is Dependency Testing because the Test Methods were skipped and did not fail. You can get more information about Dependencies on TestNG website. testng.org/doc/documentation-main.html

Click 5.7 – Dependencies and it takes you straight to that section. Next in Chapter 6.2, we will discuss the Depends On Methods Attribute which helps us with Dependency Testing.

Chapter 6.2 – Depends On Methods Attribute

Introduction

Welcome To Chapter 6.2, The Depends On Methods Attribute. In this chapter, we will cover execute Test Methods without the Depends On Methods Attribute then execute Test Methods with the Depends On Methods Attribute and execute an xml file to exclude a Test Method.

Execute Without Depends On Methods

In this example, we have 7 Test Methods. I placed set up Chrome in a Test Annotation. We want to test, we can set up Chrome. In addition, the other Test Methods are Open Orange HRM, Sign In, Search User,

Search Employee, Search Candidate, and Sign Out. This example does not have a dependsOnMethods attribute. Here's the Test Steps with our Test Application. After signing in we will go to the Admin tab and Search for a User, go to the PIM tab and search for an employee, go to the Recruitment tab and search for a candidate then sign out.

Let's run our Test Script and see what happens. This site can't be reached. Refused to connect. Go to the Console. There's a cascade of failures. A cascade failure is when 1 failure force the remaining test in the suite to fail. We see 7 runs and 6 of those runs are Failures. Why do we have 6 Failures? We could not connect to the application to start our testing.

That's not right and the Test Report is not correct. We should have 1 Pass, 1 Failure, and 5 Skips. It should be 5 Skips because we could not open the OrangeHRM application to execute the last 5 Test Methods: Sign In, Search User, Search Employee, Search Candidate, and Sign Out. To solve this problem, we can use the Depends On Methods Attribute.

Execute With Depends On Methods

If we go to the annotations package, we can find the Depends On Methods attribute in a Configuration Annotation along with the groups and Depends On Groups attribute. However, we are going to use the Test Annotation that also have the Depends On Methods attribute, Depends On Groups attribute, and Groups attribute.

We add the attribute within parenthesis after the annotation. Lowercase d for dependsOnMethods equals. Next, we write the name of the Test Method that this Test Method depends on. Test2_OpenOrangeHRM depends on Test 1_SetUpChrome. Hover over the dependsOnMethods attribute and the description states "The list of methods this method depends on". Test3_SignIn depends on Test 2_OpenOrangeHRM and the list goes on. Test 4_Search User depends on Test 3_SignIn. There are some cases where a Test Method depend on more than one Test Method. It's not required in this Test Script but I want to show you how to add multiple Test Methods to depend on. Test 5 depends on Test 2 and Test 3. We surround the Test Methods with curly brackets and separate the Test Methods with a comma. The same for Test 6 and Test 7. Copy and Paste.

Let's Run. Now, we see 1 Failure and 5 Skips. Set Up Chrome is the only Test Method that Passed. The Results tab also shows 5 Skips.

Exclude Test Method via xml File

Exclude a Test Method using an xml File. I converted the dependsOnMethods_PASS class file into this xml file called Depends On Methods.

By default, an xml file does not include a methods tag because it's optional. For our example, I added an opening and closing methods tag then exclude a Test Method. That Test Method is test4_SearchUser.

Let's imagine, everyone in the organization know that searching for a user is not working. The defect has been reported and we will add it back to our Test Suite once the defect has been resolved. In the meantime, we don't have to keep executing that Test. Exclude it for now.

Let's Run. There are zero Failures, zero Skips, and we do not see Test 4 Search For User. Go to the Results tab, no Test 4 because it was not included at Runtime. The Depends On Methods attribute is great for small test when 1 Test Method depends on another Test Method. Next, is Chapter 6.3, groups and Depends On Groups Attribute which makes our Dependency Testing more robust and easy to scale.

Chapter 6.3 / groups and Depends On Groups Attribute

Introduction

Welcome to Chapter 6.3, Groups and Depends On Groups Attribute. In this chapter, we will discuss the Groups Attribute, Depends On Groups Attribute, and Execute Groups At Runtime using an xml File.

Groups Attribute

The Groups Attribute is a way to add any number of Test Methods to a named list. This feature is beneficial because it's an opportunity to run only those lists of Test Methods according to their group name. The group names can be a module of our application, a certain Test Type, whatever we decide. Plus, one Test Method can be a part of more than one group. For example, you can add the Sign In Test Method to a group for Smoke Test and to another group for Regression Test. Smoke Test is a Test Type that makes sure the most important functions in an application still work after a build is deployed to an environment. Regression is when a set of Test Scripts are re-executed to make sure the existing functions are still working after an application change.

In this example, test1_SetUpChrome is added to a group called initialize. The description for groups show "The list of groups this class/method belongs to". A class can also be added to a group.

Depends On Groups Attribute

The Depends On Groups Attribute is similar to the Depends On Methods attribute. Depends On Groups allow us to define a group name that our Test Method depends on. The next Test Method test2_OpenOrangeHRM depends on the initialize group. Hover over the attribute. The description states "The list of groups this method depends on. Every method member of one of these groups is guaranteed to have been invoked before this method."

Also, this Test Method is part of a group called env_ application. Let's pretend there is a different URL for each environment such as DEV Environment, QA Environment, and Production Environment. So, we have a group name for those environments.

We are not forced to add our Test Methods to a group. The remaining Test Methods do not belong to a group but only depends on a group. They depend on group env_application: test3_SignIn, test4_SearchUser, test5_SearchEmployee, test6_SearchCandidate, and test7_SignOut all depend on the same group.

Let's run. We see the application is in Spanish. It was changed. All Test Methods Pass. The Results tab also shows green for all Test Methods. Now, let's run and see what happens when there's a Failure. Could not connect to the application. There's 1 Failure and 5 Skips just like with the Depends On Methods attribute.

Execute Groups At Runtime via xml File

Execute Groups At Runtime using xml. TestNG provides a few ways to run groups at Runtime. The main way is to use an xml file but we can also use the command line or ant. Here's a fake Amazon class site with 6 Test Methods. Test 1 is Login and has been added to the smoke test group. Test 2 is Search Products and has been added to the smoke test and regression test groups. Test 3 is Place Order and it has been added to the regression test, integration test, and defect group. We have the option of being very descriptive with our group name so I added dot fix. This Test Method had a defect that should be fixed. If it is fixed then we can remove Place Order from the defect group.

Why do we have a defect group? We can add a defect group for a few reasons. When a Test Method keeps failing, we can add it to the defect group to avoid running that test because we know it's not fixed yet. On the flip side, we can execute only the defect group then verify which Test Methods are still broken.

Test 4 Send Confirmation has been added to system testing and defect.backlog group. This defect will not get fixed no time soon. Test 5 is Ship Order and has been added to the groups regression and defect.progress. Test 6 is Log Out and it has no groups.

Let's execute some scenarios for groups at runtime using the xml file – All Groups. Notice, the groups tag and run tags. The groups tag allows 2 tags. We see the run tag which list the groups that can be included or excluded. This file does not include or exclude any groups. Therefore, all groups will show up. The groups tag also allows a define tag which help us create a new group based on existing groups. Let's Run the Test Suite and we see all groups. Test Methods 1 – 6 show up and Passed.

The next xml file includes a regression group. The Console show Test Methods added to the regression group. We see other groups because those Test Methods are part of more than 1 group.

In the xml file, we can include more than 1 group at Runtime. This file includes the smoke test and system test group. There are 3 Test Methods.

Look what happens when we exclude regression. As expected, there are no Test Methods in the Console assigned to a regression group.

We have the ability to use regular expressions. In this xml file, defects have a regular expression using a dot and asterisk. Out all of those Test Methods assigned to regression. Only 1 Test Method shows up. That's because, the other Test Methods are assigned to a regression group and defect group. Therefore, it's not in the Console because defects are not included.

Run Test Methods According To Group Name

Let's recap the scenarios we covered when executing groups at Runtime. A run that has No Include and No Exclude will run all Test Methods. If we include a group like regression then only Test Methods assigned to that group will run. The same goes if we include more than 1 group. We used smoke and system. Only Test Methods assigned to those groups were executed. Excluding a group name runs all Test Methods except the Test Methods assigned to that group. Last but not least, we can use regular expressions. In this example, defect used a regular expression. If a Test Method belongs to a group that is included and excluded then it will not run because excluded wins. There were 3 Test Methods

assigned to regression but only 1 showed up. The other 2 did not show up because they were assigned to regression and defect.

So far, up to this point, all of the Test Methods were executed from the same class. However, TestNG is not limited to executing a group using 1 class. We can execute groups using more than 1 class or even a package. This xml file has 2 class tags. By the way, we write the class name using the package name, dot operator sub package name, dot operator then the class name. For example, the first class name shows com.testautomationu. That's the package name and sub package name is chp6dependencyteting then we see Amazon_FakeSite. For this class, we include the regression group.

For the last class, we include env_application. Recall this class, include test2_OpenOrangeHRM that is assigned to group env_application.

Let's Run. All of the Test Methods show up. We saw the Test Script open OrangeHRM. The Results tab reflect the Console by showing both classes.

Next in Chapter 7, we will discuss Data Driven Testing which allow us to execute 1 Test Script with more than 1 set of Test Data.

Chapter 7: Data Driven Testing

Introduction

Hello and Welcome To Chapter 7, Data Driven Testing. In this chapter, we will discuss the DataProvider Annotation, the dataProvider Attribute, and the dataProviderClass Attribute. Let's take a look at the programming code before discussing the Annotation and both Attributes.

Line 11 shows a logIn method with 3 parameters: email, password, and success. They will receive their argument values from the logInData method. The values are placed in a 2-Dimensional Array. There are 3 rows of Test Data. Row 1, Row 2, and Row 3. The 1st column of data will pass email values to String email. The 2nd column of data will pass password values to String password. The 3rd column of data will pass success values to boolean success. Statement return data help us return values to the logIn method.

After the logIn method receives the argument values then the parameters use those values. Those values are printed in the following print statement. We see email, password, and success. That's a brief overview of this Java code for parameters, arguments, and 2-Dimensional Array.

DataProvider Annotation

Now, let's discuss the DataProvider Annotation. The DataProvider Annotation returns Java objects which are values to the Test Method. We implement a DataProvider Annotation by writing @DataProvider above the method that has the Test Data. In this case, logInData has the Test Data. Next, we import the annotation. The description states "Marks a method as supplying data for a test method". The logInData method will supply data to the logIn Test Method.

The DataProvider Annotation presents 2 purposes at the same time. The 1st purpose is to pass an unlimited number of values to a Test Method. There is no restriction. The values can be any Java Data

Type. In our example, we use a String and boolean Data Type. The 2nd purpose is to allow the Test Method to be invoked with different data sets. Each set will run and have its own Test Results. In our example, we have 3 sets of data.

dataProvider Attribute

The dataProvider Attribute. The dataProvider Attribute connects the DataProvider Annotation to the Test Method. This how both methods converse with each other. Since it's an attribute of the Test Annotation, we write dataProvider with a lowercase d equals the DataProvider's method name logInData surrounded by double quotes. Let's run.

All 3 login Data sets Passed. Here's the print statements that show Email, Password, and Successful Log In as true or false. The Results tab shows all 3 data sets Passed.

We can also give the DataProvider Annotation a name. In this example, the name is login-provider. We connect the DataProvider Annotation and the Test Method by using the DataProvider name and not the method name. The attribute shows login-provider. We see the DataProvider's name is optional because it executed successfully the last time without a name. Either way is okay.

Let's quickly walk through this Selenium Code for entering each data set for email and password. Line 24 enters the email for all 3 data sets. First is TestNG@Framework.com then Joe@Doe.com, and last is Test@AutomationU.com. Joe@Doe.com is the only one that is not valid.

Line 25 enters the password. Just like with Email, a password is entered for all 3 data sets. Let's Run. Authentication Failed for Joe@Doe.com. The console shows 1 Failure. We see the print statements. The Results tab also shows 2 Passed and 1 Failure. AssertionError - The Actual & Expected Results Do Not Match expected Sign Out but found Contact Us.

dataProviderClass Attribute

The dataProviderClass Attribute allows us to separate the Test Method and DataProvider into different classes. Up to this point, our Test Method and DataProvider were located in the same class. The dataProviderClass is an attribute of the Test Annotation just like the dataProvider. Here's the DataProvider Annotation.

We see OrangeHRM and SignInDP are 2 different classes. SignInDP has the DataProvider Annotation and OrangeHRM has the Test Method. To connect these 2 classes, we make the DataProvider class static then write dataProviderClass = SignInDP.class in the Test annotation. There are 4 different data sets in the DataProvider Annotation. 2 valid data sets and 2 invalid data sets.

email and password. My bad, this should be username. ALT + SHIFT + R will change the name in all places. username and password receive their data sets at Lines 22 and 23.

Let's Run. We see 2 Failures. Not Valid / NotValid34, false and Invalid / Invalid123, false are the 2 failures. Here's the 2 Passed admin / admin123. The difference is one A is capitalized and the other a is lowercase. The Results tab shows the first and last data sets Failed. We can gather our Test Data in one place and reuse it with many Test Methods. Next is Chapter 8 Cross Browser Testing which executes one Test Script on more than one browser at the same time.

Chapter 8: Cross Browser Testing

Introduction

Hello and Welcome To Chapter 8, Cross Browser Testing. In this chapter, we will discuss the Parameter tag for the TestNG xml file, the Parameters Annotation, and the different ways to supply Test Data. Cross Browser Testing is a form of Data Driven Testing because we can drive different data sets using the parameter tag and Parameters Annotation. I'm going to send different browser names from the xml file to the Parameters Annotation.

Parameter Tag via xml File

The parameter tag specifies the name and value of a parameter. In this xml file, we have the same class name "Test Automation U" located in 3 different Test: One test is for Internet Explorer, one test is for Firefox, and the other test is for Chrome.

We have the option of placing the parameter tag within the suite level or test level. The parameter tag will get overridden at the test level if we add the tag in both places with the same parameter name. This xml file has different parameter names. We see URL and BrowserType. Why will it get overridden at the test level if the parameter name was the same? It's overridden at the test level because the test level is the closest to the class level which uses the parameter name. I decide the level to place the parameter name and value by determining if all classes need the same value. If all classes need the same value then I add the parameter tag at the suite level. If the classes require a unique value then I add the parameter tag at the test level.

All 3 classes need to access the same website. Therefore, at the suite level, we have parameter name = "URL" in parenthesis then the value of the URL is https://testautomationu.applitools.com.

Next, is the test level. The parameter name will be the same for each test but the values are going to be different. Let's start with Test On IE. The parameter name is BrowserType and the value is InternetExplorer. This test will use Internet Explorer as a browser.

Test on Firefox is next. It uses Firefox as the browser. The last test is Chrome which has a value of Chrome.

Parameters Annotation

The purpose of a Parameters Annotation is to point out how to pass parameters and which parameters to pass to the Test Method.

We are going to pass the URL and BrowserType from xml to the Parameters Annotation. Import the annotation and description states "Describes how to pass parameters to a Test Method". The value we write in this annotation must match the parameter name from the xml file. The parameter names are URL and BrowserType.

Parameters in the annotation are different from parameters in the Test Method. Parameters in the annotation are a list of parameter names to be looked up in the xml file. Parameters in the Test Method receives the values from the xml file. Therefore, one gets the parameter name and the other one gets the parameter value.

Let's walkthrough this code before we execute. If the Browser Type is Internet Explorer then IE gets setup and open. After it is open then the window maximizes and load the url. The url https://testautomationu.applitools.com is sent from the xml file. The rest are print statements to show which browser is open, print the title, get the motto, and show which browser is closed. The same process repeats for each test in the xml file.

We must run the xml file. A failure will happen if we execute from the class. We saw all 3 browsers load with no Failures. First, we see Open Internet Explorer

Test Automation University | Applitools

Your path to become a QA Automation superstar!?

Close Internet Explorer

Next, we see Open Firefox and the print statements for Firefox. Next, we see Chrome and the print statements with Chrome. The Results tab shows all 3 tests passed with their data set.

In this chapter and Chapter 7, we hard-coded our Test Data sets.

Different Ways To Supply Test Data

However, we can use different ways to supply our Test Data. Those ways are CSV File, Database, Properties File, Microsoft Excel, and Hard Code our Data. All of the ways have their Pros and Cons. Next in the final Chapter Number 9, I will wrap up Chapters 1 – 8 then provide a brief overview of additional TestNG Concepts that were not covered in our course.

Chapter 9: Summary of TestNG

Introduction

Hello and Welcome To Chapter 9, Summary of TestNG, Test Next Generation. In this chapter, I will summarize the previous chapters 1 – 8 plus discuss a few concepts that were not covered in this course.

Review Previous Chapters

In Chapter 1, Introduction To TestNG, we discussed What Is A Test Framework and What Separates TestNG From Other Frameworks. A Test Framework is a design for writing and running Test Scripts. The ability to perform so many types of testing separates TestNG from other Test Frameworks.

In Chapter 2, How To Install TestNG, we discussed the IDE's that are compatible with TestNG, the Build Tools such as Ant and Maven, the Command Line which can be used for TestNG, and the downloadable JARs.

In Chapter 3, TestNG Annotations, we discussed the Configuration Annotations and the Execution Flow of the TestNG Annotations. The Configuration Annotations start with @Before and @After which are Pre and Post Conditions for the Test Method.

In Chapter 4, Priority Attribute, we discussed the Default Execution Order For Test Methods and how the Priority Attribute helps sort our Test Methods.

In Chapter 5, TestNG Assertions, we discussed Hard Asserts and Soft Asserts. A Hard Assert stops execution after a failure and moves to the next annotation while a Soft Assert continues execution after a failure and moves to the next statement line.

In my opinion, the remaining chapter 6, 7, and 8 are important features that makes TestNG the most powerful Test Framework. Chapter 6 is Dependency Testing where we discussed 3 Attributes. The Depends On Methods attribute, Groups Attribute, and Depends On Groups Attribute.

In Chapter 7, we discussed Data Driven Testing. This type of testing allows us to run 1 Test Script with an unlimited number of data sets using the DataProvider Annotation, dataProvider Attribute, and dataProviderClass Attribute.

In Chapter 8, we discussed Cross Browser Testing. Cross Browser Testing is a form of Data Driven Testing because it can use different data sets from the xml file. The xml file has a parameter tag that contains a name and value that supplies information to the Parameters Annotation. We also discussed different ways to supply test data.

That wraps up Chapters 1 – 8.

Additional Concepts

Here's some additional TestNG concepts that were not covered in this course:

- The ability to disable a Test Method by setting enable to false
- Execute a package at runtime
- Executing TestNG from the Command Prompt
- Provide optional values using the Optional Annotation
- Add Listeners which is a registration for Test Results
- Add Logs
- And the ability to view default reports
- Create custom reports
- Last but not least, multithreading which executes multiple components of a program at the same time and there's more TestNG concepts available to us

Thanks

I want to end by saying Thanks to Angie Jones. She's good, I like her and Thanks to Applitools. They gave me an opportunity to be your instructor for Test Next Generation - TestNG (A Powerful Test Framework).

You can reach me at Rex.Jones@Test4Success.org. I have a social network that provide videos on Selenium, Java, and TestNG. The videos are available on YouTube, LinkedIn, and Facebook.

I am also the author of 6 books that covers programming and automation. Some of the books are getting updated to include videos.

Thank You for watching this course and I Wish You Much Success.