Selenium with TestNG
Lecture 3

How to pass parameter from TestNG:

Passing Parameters with testng.xml

With this technique, you define the simple parameters in the testng.xml file and then reference those parameters in the source files.

Create a java class file named **LoginTest .java** in **src\test\java\UI**

```
package UI;
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.chrome.ChromeDriver;
import org.openqa.selenium.edge.EdgeDriver;
import org.openqa.selenium.firefox.FirefoxDriver;
import org.testng.annotations.Optional;
import org.testng.annotations.Parameters;
import org.testng.annotations.Test;
public class LoginTest {

		public static WebDriver driver;

		@Parameters({"browser"})
		@Test
		public void launchApplication(@Optional("chrome") String browser) {

			if(browser.equals("firefox")) {
				driver = new FirefoxDriver();
			}
			else if(browser.equals("chrome")) {
		      driver =  new ChromeDriver();
			}
			else if(browser.equals("edge")) {
				driver =  new EdgeDriver();
			}
	driver.get("https://www.saucedemo.com/v1/");
	driver.findElement(By.id("user-name")).sendKeys("standard_user");
	driver.findElement(By.id("password")).sendKeys("secret_sauce");
	driver.findElement(By.id("login-button")).click();
```

```
            driver.close();

        }
}
```

Create testng2.xml in **SeleniumTestNGGroup1\testng2.xml** to execute test case(s).

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE suite SYSTEM "https://testng.org/testng-1.0.dtd">
<suite name="Reading Browser Details from testng.xml">
 <test name="Reading Browser Details from testng.xml">
 <parameter name="browser" value="chrome"></parameter>
 <classes>
 <class name="UI.LoginTest"></class>
 </classes>
  </test> <!-- Test -->
</suite> <!-- Suite -->
```

What are TestNG Listeners:

TestNG listeners are interfaces in the TestNG framework that allow you to hook into the test execution process. They enable customization of test behavior, such as logging, reporting, or taking actions before, after, or when a test fails, passes, or is skipped.

Benefits of using TestNG Listeners with Selenium

TestNG Listeners are one of the key features of TestNG, and when used in conjunction with Selenium, they offer several benefits.

- **Enhanced Test Reporting:** By implementing listeners, you can capture and log events occurring during test execution, such as test case start, test case failure, test case success, etc.
- **Test Result Analysis:** Define custom actions to be taken when a test fails, such as capturing a screenshot, logging additional information, or sending a notification. This enables you to take immediate corrective actions and facilitates efficient debugging.

- **Test Data Manipulation:** Listeners provide hooks to modify or manipulate test data during runtime. You can use listeners to dynamically update test data, parameters, or configurations before or after each test case execution.
- **Test Execution Control:** Listeners allow you to define conditions and logic for executing or skipping tests based on specific criteria.
- **Test Parallelization:** TestNG parallel test execution and listeners play a crucial role in parallel test management. This can significantly reduce the test execution time, enabling quicker release cycles.
- **Custom Test Execution Behaviors:** The flexibility to define custom behaviors during test execution allows you to tailor the testing framework to suit your requirements and enhance test automation capabilities.

Types of TestNG Listeners in Selenium

Listeners are implemented in code via interfaces to modify TestNG behaviour. Listed below are the 8 most commonly used TestNG listeners:

1. IAnnotationTransformer
2. IExecutionListener
3. IHookable
4. IInvokedMethodListener
5. IMethodInterceptor
6. IReporter
7. ISuiteListener
8. ITestListener

IAnnotationTransformer:

```
package UI;
import org.testng.annotations.Test;
import Common.MyListener1;

public class IAnnotationTransformerDemo {
	MyListener1 obj=new MyListener1();
	@Test(invocationCount=5)

	public void ChangeInvocationCountOfMethod()
	{
```

```
        System.out.println("This method have invocation count set to 5 but at run time it shall become "+
obj.counter);
        }
        }

package Common;
import java.lang.reflect.Constructor;
import java.lang.reflect.Method;
import org.testng.IAnnotationTransformer;
import org.testng.annotations.ITestAnnotation;
public class MyListener1 implements IAnnotationTransformer {
        public int counter=3;
        @Override
        public void transform(ITestAnnotation testAnnotation, Class testClass, Constructor
testConstrutor, Method testMethod)
        {
        if (testMethod.getName().equals("ChangeInvocationCountOfMethod")) {
        System.out.println("Changing invocation for the following method: " + testMethod.getName());
        testAnnotation.setInvocationCount(counter);
        }
        }
        }


<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE suite SYSTEM "https://testng.org/testng-1.0.dtd">
<suite name="Suite">
<listeners>
<listener class-name="Common.MyListener1"></listener></listeners>
 <test name="Test">
  <classes>
    <class name="UI.IAnnotationTransformerDemo"/>
  </classes>
 </test> <!-- Test -->
</suite> <!-- Suite →
```

IExecutionListener:

As the name suggests, it monitors the beginning and end of TestNG execution. This listener is mainly used to start/stop the server while starting or ending code execution.

- onExecutionStart() – invoked before TestNG starts executing the suites
- onExecutionFinish() – invoked after all TestNG suites have finished execution

```java
package UI;
import org.testng.annotations.Test;

public class IExecutionListenerDemo {
	@Test

	public void method1()
	{
	System.out.println("this method is method 1");
	}
	@Test

	public void method2()
	{
	System.out.println("this method is method 2");
	}
	@Test

	public void method3()
	{
	System.out.println("this method is method 3");
	}
	@Test

	public void method4()
	{
	System.out.println("this method is method 4");
	}
	@Test

	public void method5()
	{
	System.out.println("this method is method 5");
	}
	}
```

```java
package Common;
import java.sql.Time;
import org.testng.IExecutionListener;
public class MyListener2 implements IExecutionListener{

	@Override
	public void onExecutionFinish() {
	long endTime= System.currentTimeMillis();
	System.out.println("Inform all the suite have finished execution at"+ endTime);
	}
	@Override
```

```
        public void onExecutionStart() {
        long startTime= System.currentTimeMillis();
        System.out.println("Inform all the suite have started execution at"+ startTime);
        }
        }
```

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE suite SYSTEM "https://testng.org/testng-1.0.dtd">
<suite name="Suite">
<listeners>
<listener class-name="Common.MyListener2"></listener></listeners>
 <test thread-count="5" name="Test">
   <classes>
     <class name="UI.IExecutionListenerDemo"/>
   </classes>
 </test> <!-- Test -->
</suite> <!-- Suite -->
```

ITestListener

This is the most frequently used TestNG listener. ITestListener is an interface implemented in the class, and that class overrides the ITestListener-defined methods. The ITestListener listens to the desired events and executes the methods accordingly.

- **onStart():** invoked after test class is instantiated and before execution of any testNG method.
- **onTestSuccess():** invoked on the success of a test
- **onTestFailure():** invoked on the failure of a test
- **onTestSkipped():** invoked when a test is skipped
- **onTestFailedButWithinSuccessPercentage():** invoked whenever a method fails but within the defined success percentage
- **onFinish():** invoked after all tests of a class are executedThe above-mentioned methods use the parameters ITestContext and ITestResult. The ITestContext is a class that contains information about the test run. The ITestResult is an interface that defines the result of the test.

ItestListenerDemo

```
package UI;
```

```
import org.testng.Assert;
import org.testng.SkipException;
import org.testng.annotations.Test;
import org.testng.asserts.SoftAssert;

public class ItestListenerDemo {
	int i=0;
	@Test

	public void testMethod1()
	{
	System.out.println("This method will pass and will invoke the onTestSuccess method of ITestlistener");
	int i=10;
	Assert.assertEquals(i, 10);
	}
	@Test

	public void testMethod2()
	{
	System.out.println("This method will fail and will invoke the onTestFailure method of ITestlistener");
	int i=10;
	Assert.assertEquals(i, 11);
	}
	@Test

	public void testMethod3()
	{
	System.out.println("This method will skip and will invoke the onTestSkipped method of ITestlistener");
	throw new SkipException("Skipping this test case.");
	}
	@Test(successPercentage=50, invocationCount=5)
```

```
        public void testMethod4()
        {
        i++;
        System.out.println("Test Failed But Within Success Percentage Test Method, invocation count: "
+ i);
        if (i == 1 || i == 2) {
        System.out.println("this will be Failed");
        Assert.assertEquals(i, 100);
        }
        }
        }
```

Listeners

```
package Common;

import org.testng.ITestContext;
import org.testng.ITestListener;
import org.testng.ITestResult;

public class Listeners implements ITestListener{

        @Override
        public void onFinish(ITestContext contextFinish) {
        System.out.println("onFinish method finished");

        }

        @Override
        public void onStart(ITestContext contextStart) {
        System.out.println("onStart method started");
        }

        @Override
        public void onTestFailedButWithinSuccessPercentage(ITestResult result) {
        System.out.println("Method failed with certain success percentage: "+ result.getName());

        }

        @Override
        public void onTestFailure(ITestResult result) {
```

```
System.out.println("Method failed: "+ result.getName());

}

@Override
public void onTestSkipped(ITestResult result) {
System.out.println("Method skipped: "+ result.getName());

}

@Override
public void onTestStart(ITestResult result) {
System.out.println("Method started: "+ result.getName());

}

@Override
public void onTestSuccess(ITestResult result) {
System.out.println("Method passed: "+ result.getName());

}

}
```

testnglistener.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE suite SYSTEM "https://testng.org/testng-1.0.dtd">
<suite name="Suite">
<listeners>
<listener class-name="Common.Listeners"></listener>
</listeners>
 <test thread-count="5" name="Test">
   <classes>
     <class name="UI.ItestListenerDemo"/>
   </classes>
 </test> <!-- Test -->
</suite> <!-- Suite -->
```

How to Capture ScreenShot for Failed Tests

To capture screenshots for failed test cases in TestNG, you can implement an **ITestListener** and override its **onTestFailure** method. This method will be triggered whenever a test fails, allowing you to take a screenshot using Selenium's

getScreenshotAs() method and save it to a file. You'll also need to add a dependency to your project for TestNG and Selenium.

FacebookScreenshot.java

```
package UI;
	import java.io.File;
	import library.Utility;
	import org.openqa.selenium.io.FileHandler;
	import org.openqa.selenium.By;
	import org.openqa.selenium.OutputType;
	import org.openqa.selenium.TakesScreenshot;
	import org.openqa.selenium.WebDriver;
	import org.openqa.selenium.chrome.ChromeDriver;
	import org.testng.ITestResult;
	import org.testng.annotations.AfterMethod;
	import org.testng.annotations.Test;

	public class FacebookScreenshot {
	// Create Webdriver reference
	WebDriver driver;
	@Test

	public void captureScreenshot() throws Exception
	{
	// Initiate Firefox browser
	driver=new ChromeDriver();
	// Maximize the browser
	driver.manage().window().maximize();
	// Pass application url
	driver.get("http://www.facebook.com");
	// Here we are forcefully passing wrong id so that it will fail our testcase
	driver.findElement(By.xpath(".//*[@id='emailasdasdas']")).sendKeys("Learn Automation");
	}
	// It will execute after every test execution
	@AfterMethod
	public void tearDown(ITestResult result)
	{
	// Here will compare if test is failing then only it will enter into if condition
	if(ITestResult.FAILURE==result.getStatus())
	{
	try
	{
	// Create refernce of TakesScreenshot
	TakesScreenshot ts=(TakesScreenshot)driver;
	// Call method to capture screenshot
```

```
		File source=ts.getScreenshotAs(OutputType.FILE);
		// Copy method  specific location here it will save all screenshot in our project home directory and
		// result.getName() will return name of test case so that screenshot name will be same
		try{
		FileHandler.copy(source, new File("./Screenshots/"+result.getName()+".png"));
		System.out.println("Screenshot taken");
		}
		finally {
			System.out.println("Test");
		}
		}
		catch (Exception e)
		{
		System.out.println("Exception while taking screenshot "+e.getMessage());
		}
		}
		// close application
		driver.quit();
		}
}
```

Utility.java

```
package library;
import java.io.File;import org.apache.commons.io.FileUtils;
import org.openqa.selenium.OutputType;
import org.openqa.selenium.TakesScreenshot;
import org.openqa.selenium.io.FileHandler;
import org.openqa.selenium.WebDriver;
public class Utility {
		public static void captureScreenshot(WebDriver driver,String screenshotName)
		{
		try
		{
		TakesScreenshot ts=(TakesScreenshot)driver;
		File source=ts.getScreenshotAs(OutputType.FILE);
		FileHandler.copy(source, new File("./Screenshots/"+screenshotName+".png"));
		System.out.println("Screenshot taken");
		}
		catch (Exception e)
		{
		System.out.println("Exception while taking screenshot "+e.getMessage());
		}
		}
		}
```