Day 6 :

TestNG: Data provider.

What are DataProviders in TestNG?

Data Providers helps us to achieve Data Driven testing.

IT helps us to execute one test case multiple times with multiple data.

The *DataProviders* in TestNG are another way to pass the parameters in the test function, the other one being TestNG parameters. *DataProviders* pass different values to the *TestNG Test Case* in a single execution and in the form of *TestNG Annotations*.

It is a part of the inbuilt TestNG data-driven testing for which TestNG is quite popular. *DataProviders* help in passing the parameters in different ways.

This helps us to achieve :

1. Code Optimization.
2. It also saves a lot of time.
3. Helps us to achieve Data Driven Testing.

DataProvider Syntax:

The *TestNG DataProvider* is used in the following manner:

```
@DataProvider (name = "name_of_dataprovider")
```

```
public Object[][] dpMethod() {
        return new Object [][] { values}
}
```

How DataProvider works?

Data provider is an annotation available in TestNg.

It allows testers to execute a single test method with multiple sets of data.

We have to write the test method only once and mention the Data provider .

This will execute the test method multiple times with multiple sets of data provided.

Steps to Implement Data provider :

Step 1 : Create a 2 Dimensional Object array and mention the size of it wrt rows and column size of your Excel sheet.

Step 2 : Create 2 for loops (1 for Rows and 1 for Columns) and get the cell value and store it in a 2d array.

Step 3 : Return the Object array

```
public class DataExtract {

    // In this class we are getting the data from excel and
storing it all into a 2 Dimensional object array
```

```
     public Object[][] getdatafromexcel() throws 
EncryptedDocumentException, IOException {



          // Reading our Excel File through FileInputStream


          FileInputStream fis = new FileInputStream("C:\\
Users\\Dell\\Documents\\Moolyaworkspace\\Selenium_Demo\\src\\
main\\resources\\testdata.xlsx");



          Workbook book = WorkbookFactory.create(fis);



          Sheet sh = book.getSheet("Sheet1");


          // next step is get last row number and last coloumn
number


          int cellsize = sh.getRow(0).getLastCellNum();


          int rowcount = sh.getLastRowNum();



          // Next step - Create/Define a 2 dimensional Object 
array. - why? - To store the data that we are getting from 
excel.


          Object[][] obj= new Object[rowcount][cellsize];
          // Next step -define loops for storing the values 
from excel to your 2d array.

          // Here 2 loops are used so that first we keep the 
row value same and iterate over coloums and

          // do the same for all the rows


          for(int i=0;i<rowcount;i++) // this for loop is for 
Rows.

          {

               for(int j=0;j<cellsize;j++) { // this for loop 
is for Coloumns


-


obj[i][j] = sh.getRow(i+1).getCell(j).getStringCellValue();


               }




          }


          return obj;




     }



}
```

```
        @DataProvider(name = "getdata")
        public Object[][] getdata() throws EncryptedDocumentException, IOException {
        DataExtract get = new DataExtract();

        Object[][] data = get.getdatafromexcel();// calling the get data from excel and storing

        return data;// we return this data to the test
        }


        @BeforeMethod
        public void setup() {

        System.setProperty("webdriver.chrome.driver", "C:\\Users\\Dell\\Downloads\\
chromedriver120_\\chromedriver-win64\\chromedriver.exe");

        driver = new ChromeDriver();

        driver.get("https://the-internet.herokuapp.com/login");

        System.out.println("URL loaded successfully");
        }


        // The data returned from data provider is caught by test method , we enable this by
mentioning data provider in the test parameter.
        @Test(dataProvider = "getdata")

        public void logintest(String uname,String pass) throws InterruptedException {

        // here the data from data provider are automatically stored in the string uname and pass ,
        // without we storing it explicitly.
        //This is taken care by the data provider.

        WebElement username = driver.findElement(By.cssSelector("#username"));
        WebElement password = driver.findElement(By.cssSelector("input#password"));

        Thread.sleep(2000);

        // Entering the data into webelements from data provider(dataprovider to string variable ,
string to these webelements)
        username.sendKeys(uname);
        password.sendKeys(pass);



        Thread.sleep(2000);
        }
```

Note : For TestNg executing Data provider, always use @BeforeMethod and @AfterMethod for before and after test Implementation.

If we use @BeforeTest and @AfterTest then the Data provider won't work properly, the data entry will happen before closing the browser.

Groups

We are going to group our test cases into – Smoke, Regression, sanity, Functional

TestNG Groups with Example

We use groups in **Testng** when,

- We don't want to define test methods separately in different classes (depending upon functionality) and
- At the same time I want to ignore (not execute) some test cases as if they do not exist in the code.
- So to carry out this we have to Group them. This is done by using the "include" and "exclude" mechanisms supported in testNG.

Steps to generate testng.xml file –

1) Right Click on the class and select Convert to Testng.

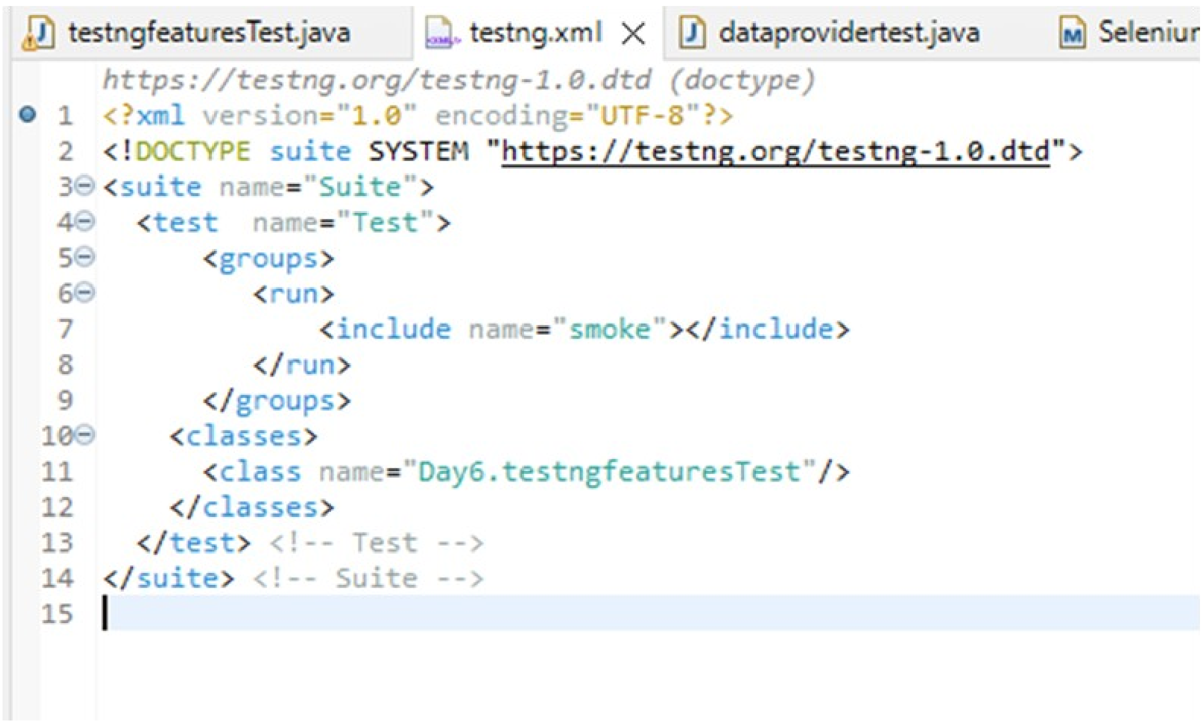
In the example below, we have shown the syntax of how to use groups in the XML file.

```
@Test (groups = { "smoke"})
```

Customize your XML to pick the mentioned group from the test classes. Below mentioned is the syntax of how to declare groups in an XML file e.g.

```
<groups>

   <run>

    <include name="bonding" />

   </run>

  </groups>
```

This is how our testng.xml should look like for using groups



```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE suite SYSTEM "https://testng.org/testng-1.0.dtd">
<suite name="Suite">
  <test  name="Test">
      <groups>
         <run>
            <include name="smoke"></include>
         </run>
      </groups>
    <classes>
      <class name="Day6.testngfeaturesTest"/>
    </classes>
  </test> <!-- Test -->
</suite> <!-- Suite -->
```

And this is how we should add groups parameter to out testNg tests.

Priority

What Is Prioritisation In TestNG?

Prioritisation in TestNG is a way to provide a sequence to the methods so that they do not run out of order. Since alphabetically running test cases in TestNG have no logical sequence (*concerning the tests and code*), providing priority to these test cases helps us managing our tests' execution.

Priority in TestNG test cases is a parameter with attribute value as "priority".

How to give Priority in TestNG test?

The following is the syntax for allocating a priority to a test case method.

```
@Test (priority = 1)
public void func(){
   //test code
}
```

In the OpenBrowser method, I am trying to open the browser and enter the URL "www.demoqa.com." The "CloseBrowser" method, however, is used to close the driver. The priorities set are 0 for OpenBrowser and 1 for CloseBrowser, so I expect the OpenBrowser method to run first.

As expected, the OpenBrowser method ran first because of a lower priority. Had I not declared the priority here, it would have run alphabetically, i.e., CloseBrowser first and then OpenBrowser.

Parallel Testing

What is Parallel Execution in TestNG?

Parallel testing is a process where multiple tests are executed simultaneously/in parallel in different thread processes.

For each browser instance, it is considering as one thread.

With respect to Selenium and TestNG, it allows you to execute multiple tests on different browsers, devices, environments in parallel and at the same time, instead of running it sequentially.

The main purpose of running tests in parallel mode is to **reduce execution time** and do **maximum environment coverage** (browsers/devices/environment) in less time.

Suppose, for an application you need to execute a sanity automation suite of 50 test cases in **Chrome** and **Firefox** browser. If you go with the traditional sequential flow, you need to execute the suite for Chrome browser first which would take 1 hr and then you need to execute for Firefox browser which takes another 1 hr.

Total time taken sequentially - 2 hours.

Time is Money.

So, in total you would need 2 hrs to test in both the browsers.

But By using a parallel mechanism, you can run simultaneously for both the browsers in just 1 hr thereby reducing the execution time by 50%.

Steps to implement Parallel testing in TestNG :

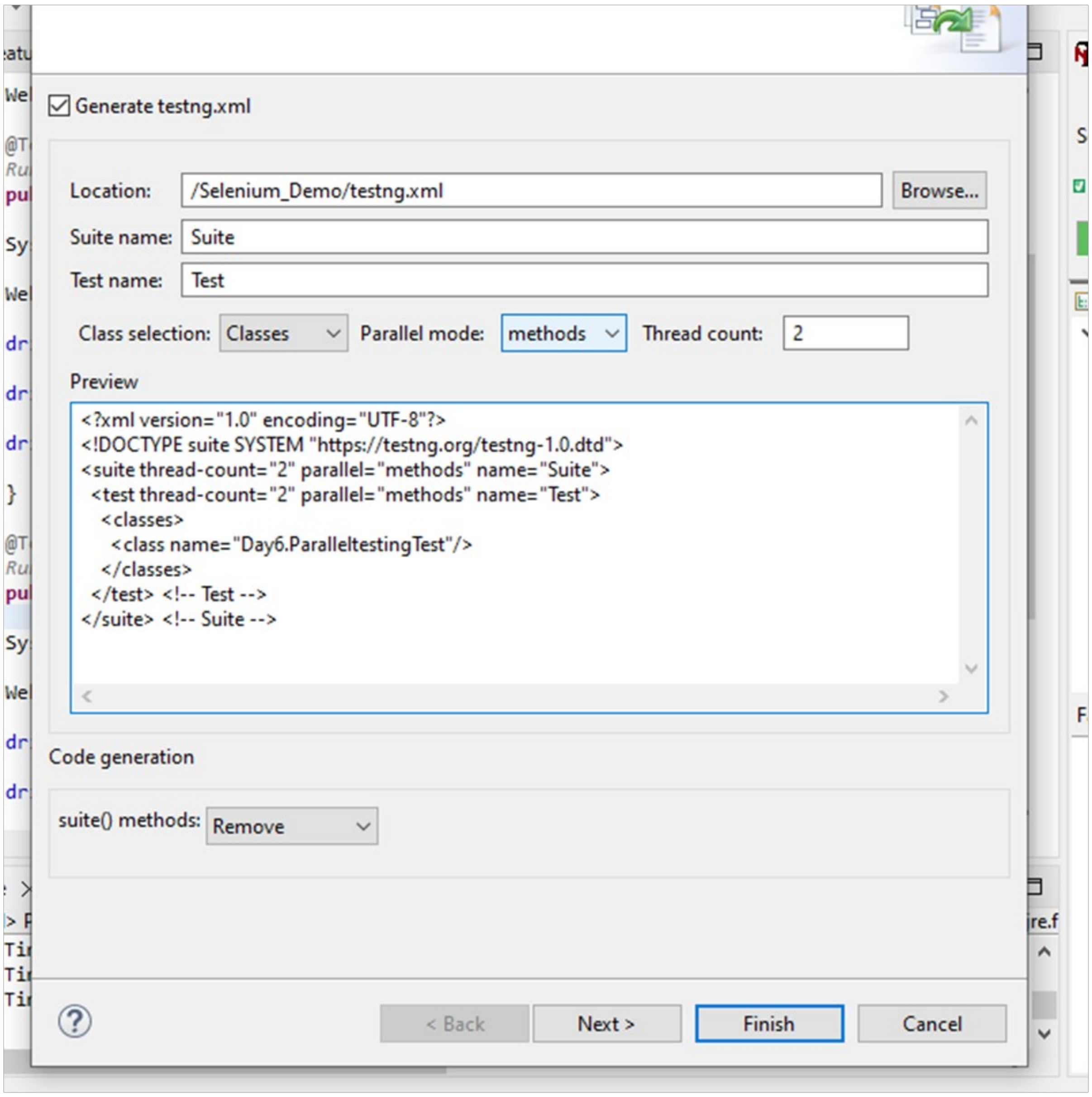
To do so you need to first create a **testing.xml** file and add a parallel attribute for the test suite with value as **methods**.

Executing Parallel Test Methods in TestNG

Step 1 To create a **testing.xml** file, right click on the **ParallelTest** class and select **TestNG** >> **Convert To TestNG**.

Step 2 You may select **Parallel mode** and **ThreadCount** value of your choice while creating the **testing.xml** file or you may update it later as per the requirement change. I have selected **Parallel mode** as **methods** and **ThreadCount** as **2**.

This is how our testng.xml file should look like :



```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE suite SYSTEM "https://testng.org/testng-1.0.dtd">
<suite thread-count="2" parallel="methods" name="Suite">
  <test thread-count="2" parallel="methods" name="Test">
    <classes>
      <class name="Day6.ParalleltestingTest"/>
    </classes>
  </test> <!-- Test -->
</suite> <!-- Suite -->
```

Step 3 Right click on the testing.xml file and select “Run As” -> “TestNG Suite”. Observe the time taken to execute both the methods in parallel mode.

```
public class ParalleltestingTest {


    WebDriver driver;
    WebDriver driver1;

    @Test
    public void testChrome() throws InterruptedException {

    System.out.println("The thread ID for Chrome is "+
    Thread.currentThread().getId());

    WebDriverManager.chromedriver().setup();

    driver = new ChromeDriver();

    driver.get("https://www.bstackdemo.com/");

    driver.manage().window().maximize();

    Thread.sleep(2000);

    }

    @Test
    public void testFirefox() throws InterruptedException {
      System.out.println("The thread ID for Firefox is "+
Thread.currentThread().getId());

      WebDriverManager.firefoxdriver().setup();

      driver1 = new FirefoxDriver();

      driver1.get("https://www.bstackdemo.com/");

      driver1.manage().window().maximize();

      Thread.sleep(2000);

      }

      @AfterClass
      public void teardown() {

            driver.quit(); // It will close all the
windows/tabs/ browser present in the driver instance
            driver1.quit();
      }
```

Programs :

```
package Day6;
public class ObjectArray2D {
	public static void main(String[] args) {
		int arr[] = new int[3]; // 1 D array

		// 2 Day Object Array is used when we want to interact with data that is having
		// rows and coloumns

		int rowsize = 2;
		int cellsize =2;

		Object[][] data = new Object[rowsize][cellsize];

		data[0][0] = 4;
		data[0][1] = 2;
		data[1][0] = 1;
		data[1][1] = 7;

		for (int i = 0; i < rowsize; i++) {

			for(int j=0;j<cellsize;j++) {



			System.out.println(data[i][j]);

		}

		}



	}
}
```

```
package Day6;


import org.openqa.selenium.By;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.chrome.ChromeDriver;

import org.openqa.selenium.firefox.FirefoxDriver;

import org.openqa.selenium.firefox.GeckoDriverInfo;

import org.testng.annotations.Test;


import io.github.bonigarcia.wdm.WebDriverManager;


public class ParallelTesting {


    // We are going to write code to exdcute a test in chrome

    // one more test in firefox browser

    //goal is to execute both these tests parallely.


    @Test

    public void chrometest() throws InterruptedException {


        WebDriverManager.chromedriver().setup();
```

```
                WebDriver driver = new ChromeDriver();

                driver.get("https://practicetestautomation.com/practice-test-
login/");

                driver.findElement(By.name("username")).sendKeys("karan");

                Thread.sleep(2000);

                driver.findElement(By.name("password")).sendKeys("arjun");

                Thread.sleep(2000);

                driver.findElement(By.id("submit")).click();

                driver.close();

        }

        @Test

        public void firefoxtest() throws InterruptedException {
}



                WebDriverManager.firefoxdriver().setup();


                WebDriver driver1 = new FirefoxDriver();


                driver1.get("https://practicetestautomation.com/practice-test-
login/");


                Thread.sleep(2000);


                driver1.findElement(By.name("username")).sendKeys("karan");


                Thread.sleep(2000);




                driver1.findElement(By.name("password")).sendKeys("arjun");


                driver1.findElement(By.id("submit")).click();


                driver1.close();

        }}

package Day6;
```

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

import java.time.Duration;

import java.util.List;


import org.junit.AfterClass;

import org.openqa.selenium.By;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.WebElement;

import org.openqa.selenium.chrome.ChromeDriver;

import org.openqa.selenium.support.ui.ExpectedConditions;

import org.openqa.selenium.support.ui.Select;

import org.openqa.selenium.support.ui.WebDriverWait;

import org.testng.annotations.AfterMethod;

import org.testng.annotations.AfterTest;

import org.testng.annotations.BeforeClass;

import org.testng.annotations.BeforeMethod;

import org.testng.annotations.BeforeTest;

import org.testng.annotations.Test;


public class RahulShettyTests {
```

```
WebDriver driver;

WebDriverWait wait;




@BeforeMethod(alwaysRun = true)

public void setup() {


        driver = new ChromeDriver();


driver.get("https://rahulshettyacademy.com/AutomationPractice/");

        driver.manage().timeouts().implicitlyWait(Duration.ofSeconds(5));

        driver.manage().window().maximize();


        wait = new WebDriverWait(driver,Duration.ofSeconds(10));


}



@Test(groups = "Smoke", priority = 0)

public void radiotest() throws InterruptedException {
		WebElement radio =
driver.findElement(By.cssSelector("input[value='radio2']"));

		wait.until(ExpectedConditions.elementToBeClickable(radio));

		radio.click();

		System.out.println(radio.isSelected());

		Thread.sleep(2000);

	}



	@Test(groups = "Smoke",priority = 1)
	public void checkboxtest() throws InterruptedException {
		WebElement checkbox1 =
driver.findElement(By.id("checkBoxOption1"));



	wait.until(ExpectedConditions.elementToBeClickable(checkbox1));
```

```
                System.out.println("my checkbox status before clicking "+checkbox1.isSelected());

                checkbox1.click();

                System.out.println("my checkbox status after clicking "+checkbox1.isSelected());

                Thread.sleep(2000);

        }

        @Test(groups = "Regression",priority = 3)
        public void textboxtest() throws InterruptedException {

wait.until(ExpectedConditions.visibilityOf(driver.findElement(By.id("autocomplete"))));
```

```
                    driver.findElement(By.id("autocomplete")).sendKeys("Value
entered");

                    Thread.sleep(2000);

          }

          @Test(groups = "Functional",priority = 4)
          public void multiplecheckstest() throws InterruptedException {

                     List<WebElement> checkboxes =
driver.findElements(By.xpath("//input[@type='checkbox']"));

                     for (WebElement check : checkboxes) {

                               check.click();

                    }
```

```
                Thread.sleep(2000);



        }



        @Test(groups = "Regression",priority = 2)

        public void dropdowntest() throws InterruptedException {


            WebElement dropdown = driver.findElement(By.id("dropdown-class-
    example"));


                wait.until(ExpectedConditions.visibilityOf(dropdown));


            Select s = new Select(dropdown);


            s.selectByValue("option3");


            s.selectByVisibleText("Option1");
                Thread.sleep(2000);
        }

        @AfterMethod(alwaysRun = true)

        public void teardown() {

            driver.quit();

            driver.quit();

        }

    }
```