Lab III: Web Application Testing
TestNG and Selenium

Software Testing and QA (COE891)

Week 6 and Week 7

1 Lab Objectives

- Checking for functionality, usability, security, compatibility, performance of the web application or website.
- Validating web applications across different browsers and platforms by Selenium.
- To overcome the disadvantages of JUnit and making end-to-end testing easily by TestNG.
- Generating a proper report by TestNG, and you can easily come to know how many test cases are passed, failed, and skipped. You can execute the failed test cases separately.

2 TestNG

TestNG is an automation testing framework in which NG stands for "Next Generation". TestNG is inspired from JUnit which uses the annotations (@). TestNG overcomes the disadvantages of JUnit and is designed to make end-to-end testing easy. Using TestNG, you can generate a proper report, and you can easily come to know how many test cases are passed, failed, and skipped. You can execute the failed test cases separately.

For example, suppose you have five test cases, one method is written for each test case (Assume that the program is written using the main method without using TestNG). When you run this program first, three methods are executed successfully, and the fourth method is failed. Then correct the errors present in the fourth method, now you want to run only fourth method because first three methods are anyway executed successfully. This is not possible without using TestNG.

TestNG in Selenium provides an option, i.e., testng-failed.xml file in test-output folder. If you want to run only failed test cases means you run this XML file. It will execute only failed test cases.

3 Using TestNG with Selenium

Default Selenium tests do not generate a proper format for the test results. Using TestNG in Selenium, we can generate test results. Here are the key features of Selenium TestNG:

- Generate the report in a proper format including a number of test cases runs.
- The number of test cases passed.
- The number of test cases failed.
- The number of test cases skipped.

Multiple test cases can be grouped more easily by converting them into testng.xml file. In which you can make priorities which test case should be executed first. The same test case can be executed multiple times without loops just by using keyword called 'invocation count.'

Using testng, you can execute multiple test cases on multiple browsers, i.e., cross browser testing. The TestNG framework can be easily integrated with tools like TestNG Maven, Jenkins, and so on.

Also, annotations used in the testing are very easy to understand like: @BeforeMethod, @AfterMethod, @BeforeTest, and @AfterTest. WebDriver has no native mechanism for generating reports. TestNG can generate the report in a readable format like the one shown below. TestNG simplifies the way the tests are coded. There is no more need for a static main method in our tests. The sequence of actions is regulated by easy-to-understand annotations that do not require methods to be static.

Uncaught exceptions are automatically handled by TestNG without terminating the test prematurely. These exceptions are reported as failed steps in the report.

4 Advantages of TestNG over JUnit

There are three major advantages of TestNG over JUnit:

1. Annotations are easier to understand.
2. Test cases can be grouped more easily.
3. Parallel testing is possible.

Annotations in TestNG are lines of code that can control how the method below them will be executed. They are always preceded by the @ symbol.

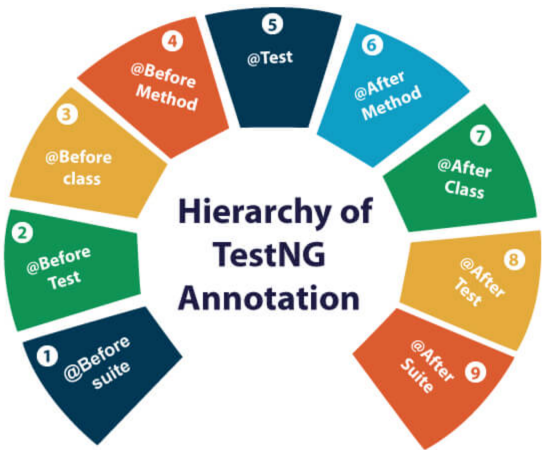

Figure 1

5 Installing TestNG and Selenium Dependencies in Eclipse

- **Step 1: Download Selenium**
  1. Go to: `https://www.selenium.dev/downloads/`
  2. Download: **Java 4.28.1** (January 23, 2025) and unzip the folder.
  3. **No need to download:** SLF4J Jar File and LogBack Jar File.
  4. **Selenium 4.28+ includes Selenium Manager**, which automatically downloads and configures WebDriver. You do not need to manually download WebDriver unless working in an offline environment or behind a firewall.
- **Step 2: Open Eclipse**
  1. Open **Eclipse Java 2023** (found under `Applications → Programming → Eclipse Java 2023`).

- **Step 3: Create a Project in Eclipse**
  1. Click `File → New → Java Project`.
  2. Uncheck `Use Default Location` and select your desired workspace.
     - Create a folder for each project under the workspace directory and include the folder name in the path.
     - This ensures all project files are saved inside each folder in the working directory.
  3. Name your project and ensure the correct Java version is set:
     - **Execution Environment JRE:** JavaSE-18
     - **Note:** Higher JavaSE versions won't work.
  4. Uncheck `Create module-info.java` file.
  5. Click `Next`.

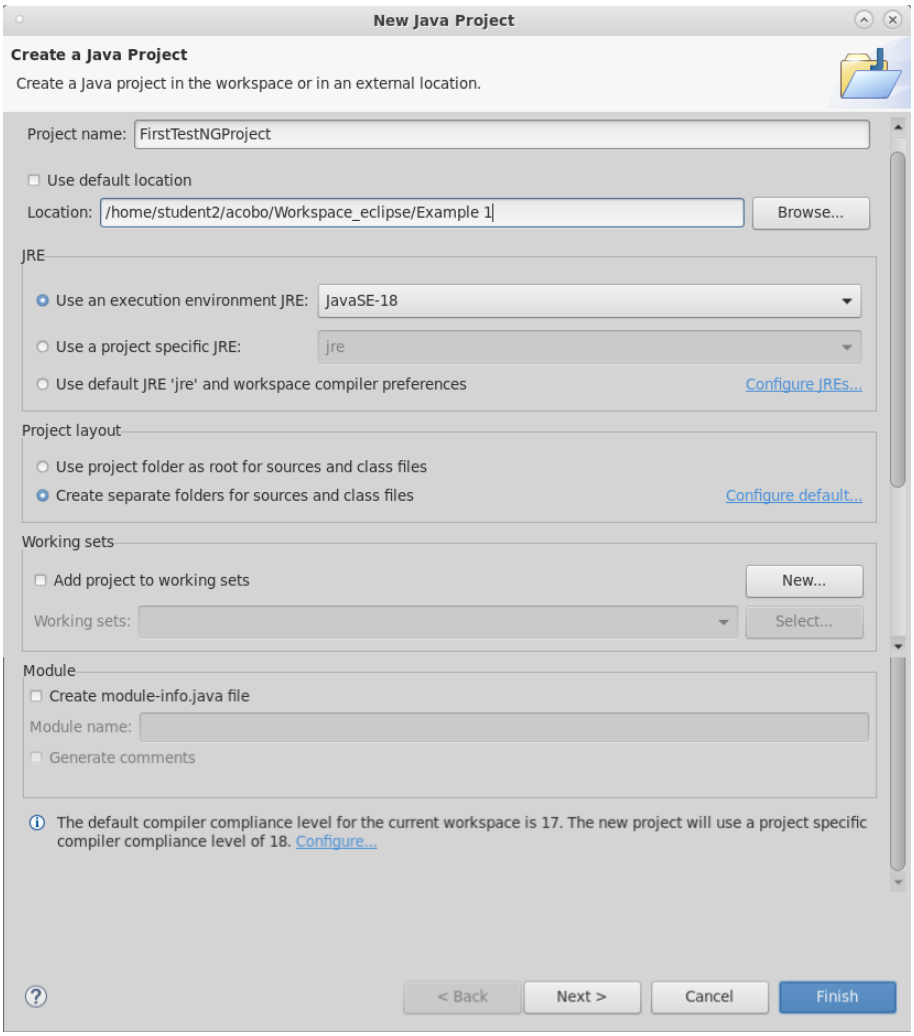

Figure 1: Creating a new Java Project in Eclipse

- **Step 4: Add TestNG to the Project**
  1. Click `Libraries → Classpath → Add Library → Select TestNG → Click Next → Click Finish`.
  2. If TestNG does not show up, follow the following section to download and add it manually (see Section 5.1).

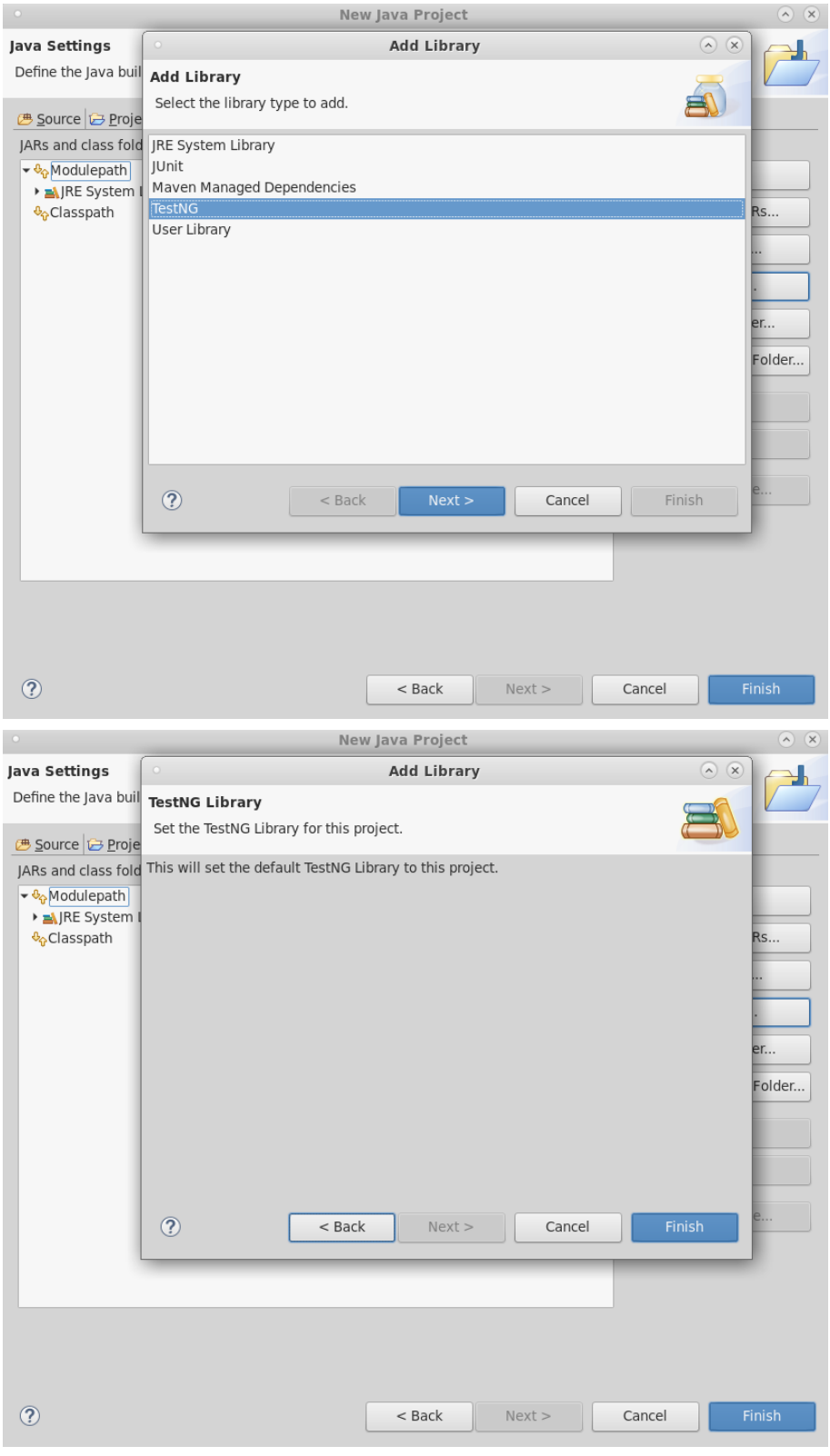

Figure 2: Adding TestNG to the project

- **Step 5: Add Selenium JAR Files**
  1. Select **Classpath** in Build Path Configuration.
  2. Click `Add External JARs` under Classpath.

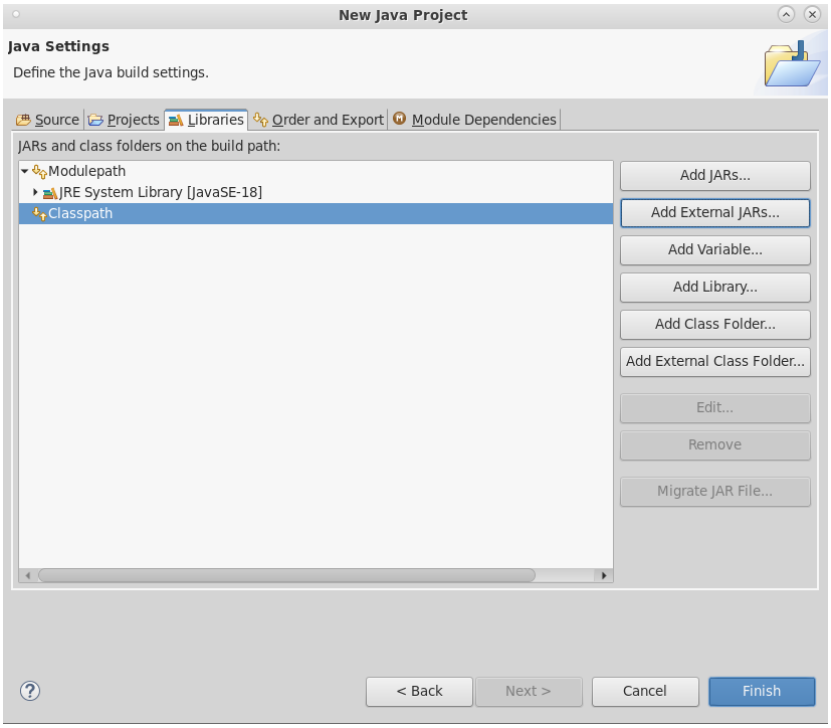

Figure 3: Adding Selenium JAR files

3. Include all Selenium JAR files found in the unzipped Selenium folder downloaded earlier.

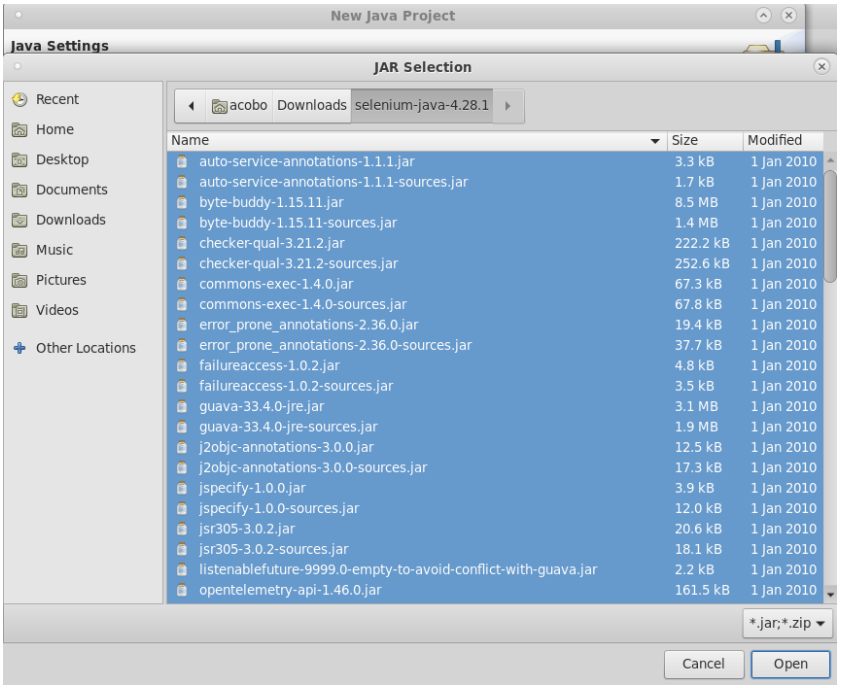

Figure 4: Adding Selenium JAR files

4. Click `Finish`.
5. Verify that **FirstTestNGProject** is visible in Eclipse's Package Explorer.

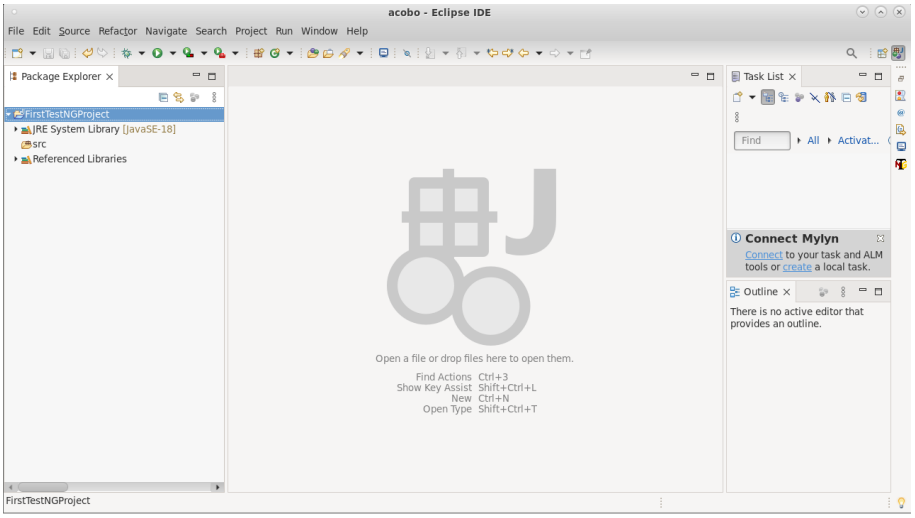

Figure 5: FirstTestNGProject

- **Step 6: Creating a TestNG File**
  1. Right-click on the `src` package folder → `New → Other...`.
  2. Click `TestNG` and select `TestNG class`.
  3. Click `Next`.

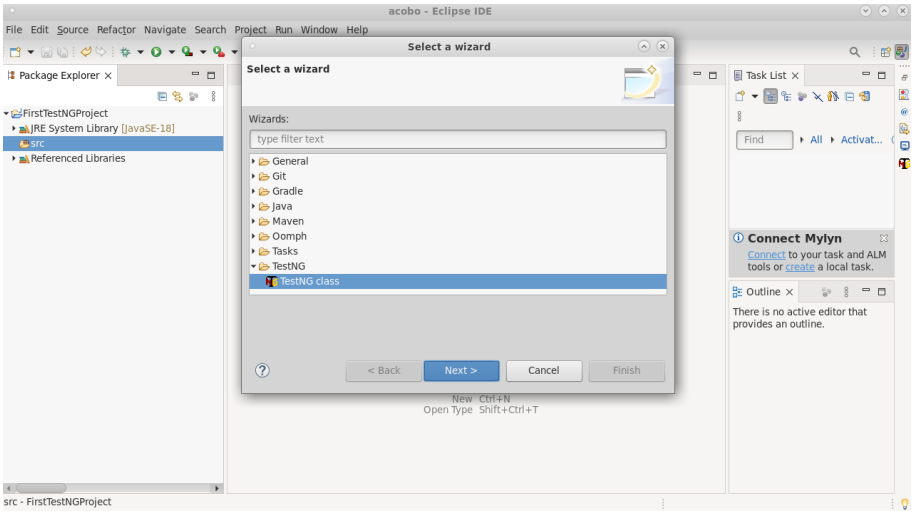

Figure 6: Creating a TestNG file in Eclipse

- Type the required values in the input boxes and click `Finish`.
  - Ensure to add `/src/` in the source folder path under the project name.
  - This ensures that the package and class are created inside the `src` folder.

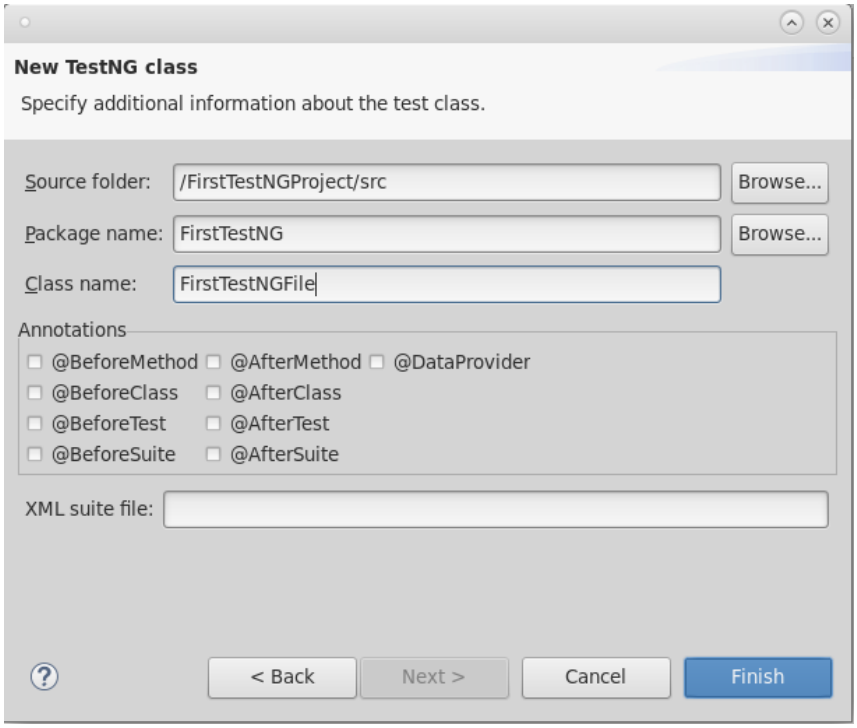

Figure 7: Configuring the TestNG file inside the src folder

5.1 Another Alternative to Create a New TestNG Test File (Manually)

Before we create a test case, we should first setup a new TestNG Project in Eclipse and name it as "FirstTestNGProject". To set up a new TestNG project:

1. Click File → New → Java Project.
2. Type "FirstTestNGProject" as the Project Name then click Next.
3. We will now start to import the TestNG Libraries onto our project. Click on the "Libraries" tab, and then "Add Library..."
4. On the Add Library dialog, choose "TestNG" and click Next.
5. Click Finish. You should notice that TestNG is included on the Libraries list.
6. We will now add the JAR files that contain the Selenium API. These files are found in the Java client driver that we downloaded from here when we were installing Selenium and Eclipse in the previous chapters. Then, navigate to where you have placed the Selenium JAR files. After adding the external JARs.
7. Click Finish and verify that our FirstTestNGProject is visible on Eclipse's Package Explorer window.

Now that we are done setting up our project in this TestNG tutorial, let us create a new TestNG file.

1. Right-click on the "src" package folder then choose New → Other...
2. Click on the TestNG inside the Java folder and select the "TestNG class" option. Click Next.
3. Type the values indicated below on the appropriate input boxes and click Finish. Notice that we have named our Java file as "FirstTestNGFile".

6 TestNG Test Case Example

Let us now create our first Test Case that will check if Mercury Tours' homepage is correct. Type your code as shown in the below TestNG Example:

```java
import org.openqa.selenium.*;
import org.openqa.selenium.firefox.FirefoxDriver;
import org.testng.Assert;
import org.testng.annotations.*;

public class firsttestngfile {
    public String baseUrl = "http://demo.guru99.com/test/newtours/";
    String driverPath = "C:\\geckodriver.exe";
    public WebDriver driver ;

  @Test
  public void verifyHomepageTitle() {
      System.out.println("launching firefox browser");
      System.setProperty("webdriver.gecko.driver", driverPath);
      driver = new FirefoxDriver();
      driver.get(baseUrl);
      String expectedTitle = "Welcome: Mercury Tours";
      String actualTitle = driver.getTitle();
      Assert.assertEquals(actualTitle, expectedTitle);
      driver.close(); }
}
```

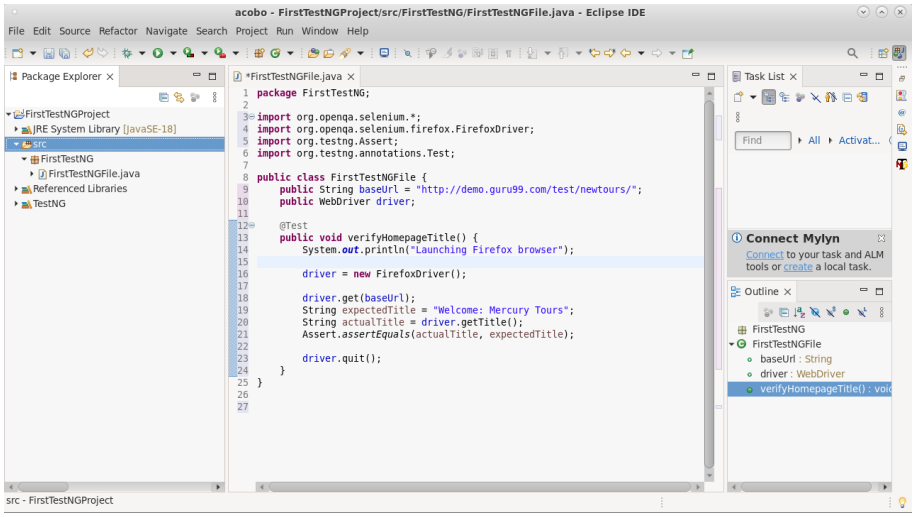

Figure 8: TestNG Test Case Code Example

1. Create a Selenium WebDriver instance.
2. Configure your browser if required(for example maximize browser, disable browser notifications etc.).
3. Navigate to the required URL (Web page).
4. Locate the HTML element.
5. Perform action on the located HTML element.
6. Verify and validate the action (concluded step).
7. Take screenshots and generate report using framework for the test cases.

Please notice the following:

- TestNG does not require you to have a main() method.
- Methods need not to be static.
- We used the @Test annotation. @Test is used to tell that the method under it is a test case. In this case, we have set the verifyHomepageTitle() method to be our test case, so we placed an '@Test' annotation above it.
- Since we use annotations in TestNG, we needed to import the package `org.testng.annotations.*`.

- We used the Assert class. The Assert class is used to conduct verification operations in TestNG. To use it, we need to import the `org.testng.Assert` package.

You may have multiple test cases (therefore, multiple @Test annotations) in a single TestNG file. This will be tackled in more detail later in the section "Annotations used in TestNG."

6.1 Running the Test

TestNG may be preinstalled on Eclipse 2023, but an **update may be required** by following below steps to ensure it runs without errors:

- Go to the Eclipse menu and navigate to Help → Eclipse Marketplace → Installed, select TestNG, and update it.
- Then, Eclipse will restart, and afterward, the program will run without errors.

To run the test, simply execute the file in Eclipse as follows:

1. Right-click on `FirstTestNGFile.java` → `Run As` → `TestNG Test`.
2. Eclipse will generate two outputs:
   - A text-based report in the **Console** window.
   - A graphical report in the **TestNG Results** window.

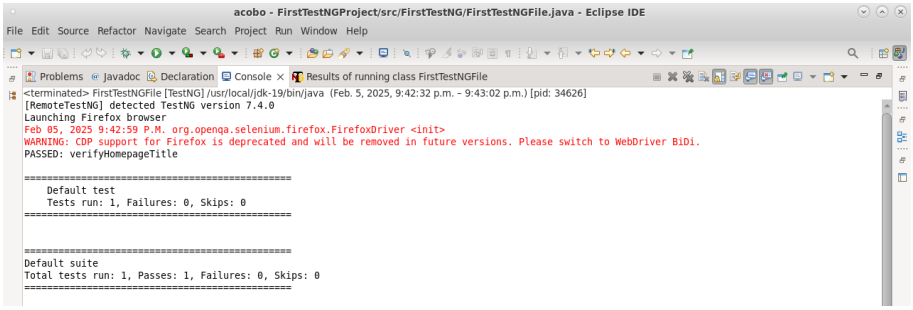

Figure 9: Running TestNG and Viewing Reports

3. TestNG can also generate reports in HTML format:
   (a) After running `FirstTestNGFile`, right-click the project name (`FirstTestNGProject`) in the **Project Explorer** window and select **Refresh**.
   (b) Notice that a `test-output` folder is created. Expand it and locate the `index.html` file.
   (c) Double-click on `index.html` to open it within Eclipse's built-in web browser.
   (d) You can refresh this page anytime after rerunning the test by pressing `F5`, just like in regular web browsers.

6.2 Convert Selenium Project to TestNG

1. Select the Java files under the package, then right-click.
2. Choose the **TestNG** option and click on **Convert to TestNG**.
3. A new window will open, prompting you to enter details such as:
   - Location
   - Suite Name
   - Test Name
   - Class Selection
   - Parallel Mode
4. Click the **Finish** button. The `testng.xml` file will be created under the Java project.

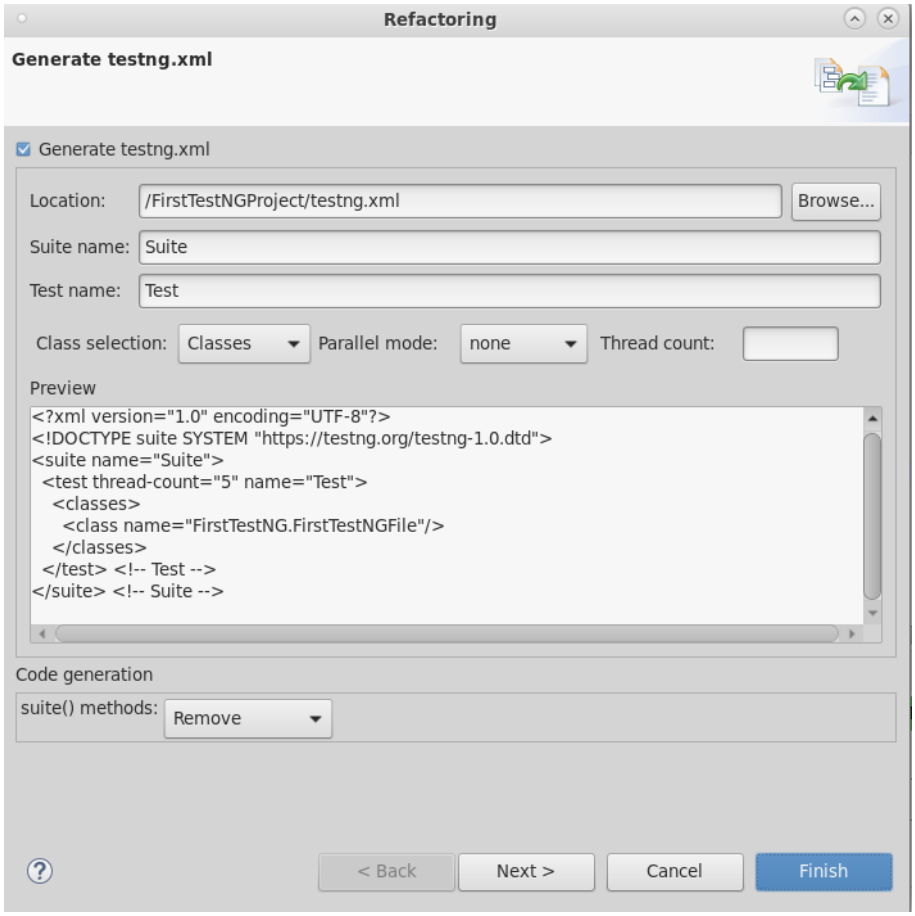

Figure 10: Converting Selenium Project to TestNG

To run the `testng.xml` file:

1. Right-click on the `testng.xml` file in the **Package Explorer**.
2. Click `Run As → TestNG Suite`.

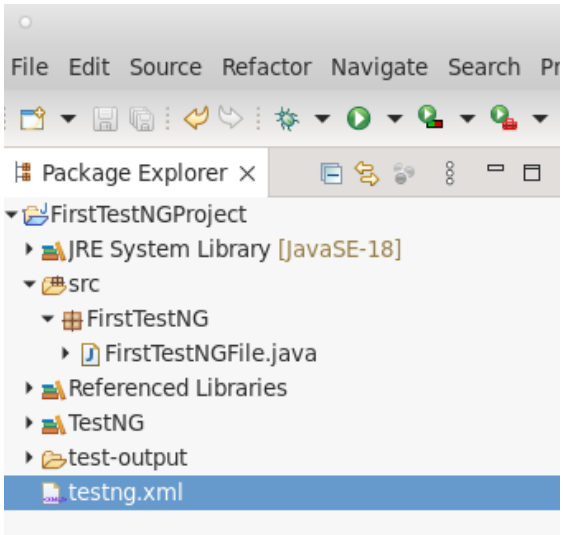

Figure 11: Running testng.xml in Eclipse

After executing the testng.xml file, it will display the result (if you have written any executable statements using `System.out.println()`).

This is one of the ways you will execute test through the eclipse.

7 Parallel Execution in TestNG

After creating xml file, in next step, we will execute the parallel test.

- thread-count: This is used for parallel execution, based on the number script. It will execute in parallel or sequential order.
- verbose: It is used to log the execution details in the console. The value should be 1-10. The log details in the console window will get more detailed and clearer as you increase the value of the verbose attribute in the testng.xml configuration file.
- name: Name of the suite.
- Parallel: To run scripts parallel, value can be "tests/classes/methods/suites". Default value is none.

Right click on the testng.xml and select run as testing, once successful you will see all the results. Here two different sessions will be generated for two different WebDriver.

```
1  import org.openqa.selenium.WebDriver;
2  import org.openqa.selenium.firefox.FirefoxDriver;
3  public class SessionHandling {
4  public static void main(String...strings ){
5      //First session of WebDriver
6      WebDriver driver = new FirefoxDriver();
7      //Goto guru99 site
8      driver.get("http://demo.guru99.com/V4/");
9      //Second session of WebDriver
10     WebDriver driver2 = new FirefoxDriver();
11     //Goto guru99 site
12     driver2.get("http://demo.guru99.com/V4/"); }
13 }
```

There are situations where you want to run multiple tests simultaneously. In such cases, the `parallel` attribute can be used. The `parallel` attribute of the `<suite>` tag in the TestNG XML file can accept four values:

- **tests**: All test cases inside the `<test>` tag of the TestNG XML file will run in parallel.
- **classes**: All test cases inside a Java class will run in parallel.
- **methods**: All methods with the `@Test` annotation will execute in parallel.
- **instances**: Test cases in the same instance will execute in parallel, but methods from two different instances will run in separate threads.

Instances will run in different thread. The attribute **thread-count** allows you to specify how many threads should be allocated for this execution.

7.1 Parallel Execution and Session Handling with Chrome

In this Example, three test cases will run parallel and fill login data in http://demo.guru99.com:

```
1  import org.openqa.selenium.WebDriver;
2  import org.openqa.selenium.chrome.ChromeDriver;
3  import org.testng.annotations.Test;
4  public class TestGuru99MultipleSession {
5      @Test
6      public void executSessionOne(){
7          //First session of WebDriver
8          System.setProperty("webdriver.chrome.driver","chromedriver.exe");
9          WebDriver driver = new ChromeDriver();
10         //Goto guru99 site
11         driver.get("http://demo.guru99.com/V4/");
12         //find user name text box and fill it
13         driver.findElement(By.name("uid")).sendKeys("Driver 1"); }
14
```

```
    @Test
    public void executeSessionTwo(){
        //Second session of WebDriver
        System.setProperty("webdriver.chrome.driver","chromedriver.exe");
        WebDriver driver = new ChromeDriver();
        //Goto guru99 site
        driver.get("http://demo.guru99.com/V4/");
        //find user name text box and fill it
        driver.findElement(By.name("uid")).sendKeys("Driver 2"); }

    @Test
    public void executSessionThree(){
        //Third session of WebDriver
        System.setProperty("webdriver.chrome.driver","chromedriver.exe");
        WebDriver driver = new ChromeDriver();
        //Goto guru99 site
        driver.get("http://demo.guru99.com/V4/");
        //find user name text box and fill it
        driver.findElement(By.name("uid")).sendKeys("Driver 3"); }
}
```

TestNG.xml:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd">
<suite name="TestSuite" thread-count="3" parallel="methods" >
<test name="testGuru">
<classes>
<class name="TestGuru99MultipleSession">
</class>
</classes>
</test>
</suite>
```

7.2 Running Parallel Tests with Firefox

The following example demonstrates running multiple WebDriver sessions in parallel using Firefox instead of Chrome.

```
package parallelExecution;

import org.openqa.selenium.WebDriver;
import org.openqa.selenium.firefox.FirefoxDriver;
import org.openqa.selenium.By;
import org.testng.annotations.Test;

public class TestGuru99MultipleSession {

    @Test
    public void executeSessionOne() {
        // First session of WebDriver
        WebDriver driver = new FirefoxDriver();
        driver.get("http://demo.guru99.com/V4/");
        driver.findElement(By.name("uid")).sendKeys("Driver 1");
        driver.quit();  // Close session after execution
    }

    @Test
    public void executeSessionTwo() {
        // Second session of WebDriver
        WebDriver driver = new FirefoxDriver();
        driver.get("http://demo.guru99.com/V4/");
        driver.findElement(By.name("uid")).sendKeys("Driver 2");
        driver.quit();  // Close session after execution
    }

    @Test
```

```
29      public void executeSessionThree() {
30          // Third session of WebDriver
31          WebDriver driver = new FirefoxDriver();
32          driver.get("http://demo.guru99.com/V4/");
33          driver.findElement(By.name("uid")).sendKeys("Driver 3");
34          driver.quit();  // Close session after execution
35      }
36  }
```

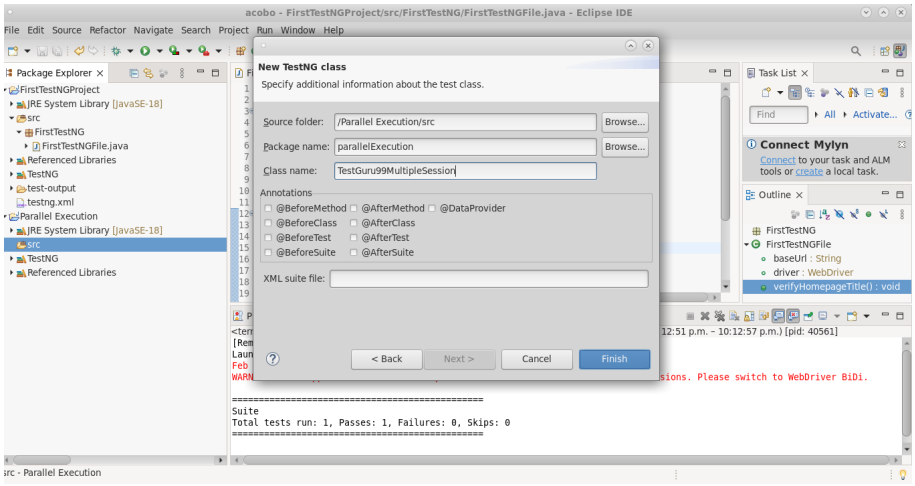

Figure 12: Creating TestNG class

7.3 TestNG.xml Configuration

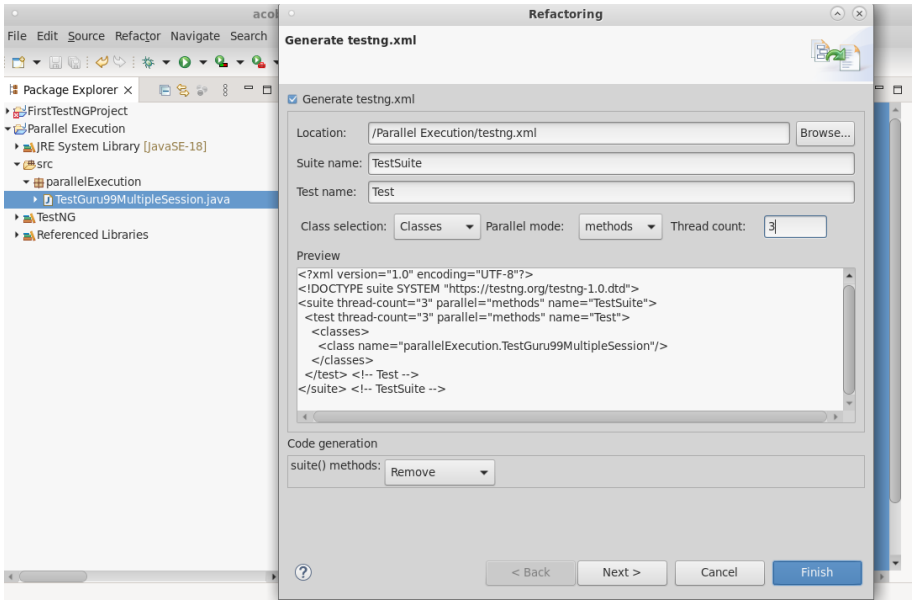

Figure 13: TestNG.xml Configuration

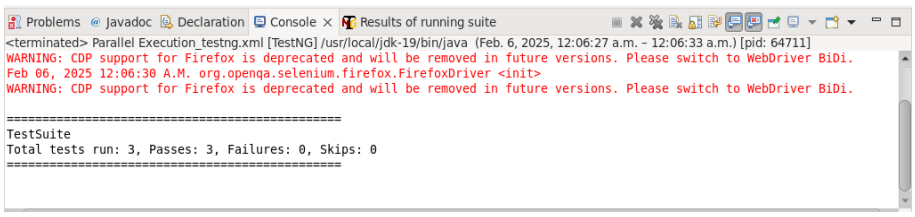

Figure 14: TestNG results

7.4 Test Case order and Dependency

You can set order and dependency of Test Case execution. Suppose you have two test cases , 'testGuru99TC1' and 'testGuru99TC2' and you want to execute test case 'testGuru99TC2' before 'testGuru99TC1'. In that case, we will use 'dependsOnMethods' attribute to make dependency and order of execution.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd">
<suite name="TestSuite" thread-count="3" parallel="methods" >
<test name="testGuru">
<classes>
<class name="TestGuru99MultipleSession">
<include value="testGuru99TC1" dependsOnMethods="testGuru99TC2"/>
<include value="testGuru99TC2"/>
</class>
</classes>
</test>
</suite>
```

8 Annotations used in TestNG

In the previous sections, you have been introduced to the @Test annotation. Now, we shall be studying more advanced annotations and their usages.

8.1 Multiple Test Cases

We can use multiple @Test annotations in a single TestNG file. By default, methods annotated by @Test are executed alphabetically. Run this code and on the generated index.html page, click "Chronological view."

8.2 Parameters

If you want the methods to be executed in a different order, use the parameter "priority". Parameters are keywords that modify the annotation's function. Parameters require you to assign a value to them. You do.this by placing a "=" next to them, and then followed by the value. Parameters are enclosed in a pair of parentheses which are placed right after the annotation like the code snippet shown below.

TestNG will execute the @Test annotation with the lowest priority value up to the largest. There is no need for your priority values to be consecutive. The TestNG HTML report will confirm that the methods were executed based on the ascending value of priority.

Aside from "priority," @Test has another parameter called "alwaysRun" which can only be set to either "true" or "false." To use two or more parameters in a single annotation, separate them with a comma such as the one shown below. $\longrightarrow$ `@Test(priority = 0, alwaysRun = true)`

8.3 @BeforeTest and @AfterTest:

@BeforeTest methods under this annotation will be executed prior to the first test case in the TestNG file. @AfterTest methods under this annotation will be executed after all test cases in the TestNG file are executed. Consider the code below.

```
import org.openqa.selenium.*;
import org.openqa.selenium.firefox.FirefoxDriver;
import org.testng.Assert;
import org.testng.annotations.*;
public class firsttestngfile {
    public String baseUrl = "http://demo.guru99.com/test/newtours/";
    String driverPath = "C:\\geckodriver.exe";
    public WebDriver driver;
```

```
    @BeforeTest
    public void launchBrowser() {
        System.out.println("launching firefox browser");
        System.setProperty("webdriver.gecko.driver", driverPath);
        driver = new FirefoxDriver();
        driver.get(baseUrl); }
    @Test
    public void verifyHomepageTitle() {
        String expectedTitle = "Welcome: Mercury Tours";
        String actualTitle = driver.getTitle();
        Assert.assertEquals(actualTitle, expectedTitle); }
    @AfterTest
    public void terminateBrowser(){
        driver.close(); }
}
```

Applying the logic presented by the table and the code above, we can predict that the sequence by which methods will be executed is:

1. launchBrowser()
2. verifyHomepageTitle()
3. terminateBrowser()

The placement of the annotation blocks can be interchanged without affecting the chronological order by which they will be executed. Let's understand with a TestNG Example and try to rearrange the annotation blocks such that your code would look similar to the one below.

```
import org.openqa.selenium.*;
import org.openqa.selenium.firefox.FirefoxDriver;
import org.testng.Assert;
import org.testng.annotations.*;
public class firsttestngfile {
    public String baseUrl = "http://demo.guru99.com/test/newtours/";
    String driverPath = "C:\\geckodriver.exe";
    public WebDriver driver;
      @AfterTest                          //Jumbled
      public void terminateBrowser(){
          driver.close(); }
      @BeforeTest                          //Jumbled
      public void launchBrowser() {
          System.out.println("launching firefox browser");
          System.setProperty("webdriver.gecko.driver", driverPath);
          driver = new FirefoxDriver();
          driver.get(baseUrl); }
      @Test                               //Jumbled
      public void verifyHomepageTitle() {
          String expectedTitle = "Welcome: Mercury Tours";
          String actualTitle = driver.getTitle();
          Assert.assertEquals(actualTitle, expectedTitle); }
}
```

8.4 @BeforeMethod and @AfterMethod:

@BeforeMethod methods under this annotation will be executed prior to each method in each test case. @AfterMethod methods under this annotation will be executed after each method in each test case.

The flow of our test would be:

1. Go to the homepage and verify its title.
2. Click REGISTER and verify the title of its target page.
3. Go back to the homepage and verify if it still has the correct title.

4. Click SUPPORT and verify the title of its target page.
5. Go back to the homepage and verify if it still has the correct title.

The code below illustrates how @BeforeMethod and @AfterMethod are used to efficiently execute the scenario mentioned above.

```
import org.openqa.selenium.*;
import org.openqa.selenium.firefox.FirefoxDriver;
import org.testng.Assert;
import org.testng.annotations.*;
public class firsttestngfile {
    public String baseUrl = "http://demo.guru99.com/test/newtours/";
    String driverPath = "C:\\geckodriver.exe";
    public WebDriver driver;
    public String expected = null;
    public String actual = null;
    @BeforeTest
    public void launchBrowser() {
        System.out.println("launching firefox browser");
        System.setProperty("webdriver.gecko.driver", driverPath);
        driver= new FirefoxDriver();
        driver.get(baseUrl); }
    @BeforeMethod
    public void verifyHomepageTitle() {
        String expectedTitle = "Welcome: Mercury Tours";
        String actualTitle = driver.getTitle();
        Assert.assertEquals(actualTitle, expectedTitle); }
    @Test(priority = 0)
    public void register() {
        driver.findElement(By.linkText("REGISTER")).click() ;
        expected = "Register: Mercury Tours";
        actual = driver.getTitle();
        Assert.assertEquals(actual, expected); }
    @Test(priority = 1)
    public void support() {
        driver.findElement(By.linkText("SUPPORT")).click() ;
        expected = "Under Construction: Mercury Tours";
        actual = driver.getTitle();
        Assert.assertEquals(actual, expected); }
    @AfterMethod
    public void goBackToHomepage ( ) {
        driver.findElement(By.linkText("Home")).click() ; }
    @AfterTest
    public void terminateBrowser(){
        driver.close(); }
}
```

After executing this test, your TestNG should report the following sequence. Simply put, @BeforeMethod should contain methods that you need to run before each test case while @AfterMethod should contain methods that you need to run after each test case. Moreover, there are other functions to find a specific element by the web driver indicated in the following:

Variation	Description	Sample
By.**className**	finds elements based on the value of the “class” attribute	findElement(By.className(“someClassName”))
By.**cssSelector**	finds elements based on the driver’s underlying CSS Selector engine	findElement(By.cssSelector(“input#email”))
By.**id**	locates elements by the value of their “id” attribute	findElement(By.id(“someId”))
By.**linkText**	finds a link element by the exact text it displays	findElement(By.linkText(“REGISTRATION”))
By.**name**	locates elements by the value of the “name” attribute	findElement(By.name(“someName”))
By.**partialLinkText**	locates elements that contain the given link text	findElement(By.partialLinkText(“REG”))
By.**tagName**	locates elements by their tag name	findElement(By.tagName(“div”))
By.**xpath**	locates elements via XPath	findElement(By.xpath(“//html/body/div/table/tbody/tr/td[2]/table/ tbody/tr[4]/td/table/tbody/tr/td[2]/table/tbody/tr[2]/td[3]/ form/table/tbody/tr[5]”))

Figure 2

8.5 Summary of TestNG Annotations

TestNG Annotation	Description
@BeforeSuite	The @BeforeSuite annotated method will run before the execution of all the test methods in the suite.
@AfterSuite	The @AfterSuite annotated method will run after the execution of all the test methods in the suite.
@BeforeTest	The @BeforeTest annotated method will be executed before the execution of all the test methods of available classes belonging to that folder.
@AfterTest	The @AfterTest annotated method will be executed after the execution of all the test methods of available classes belonging to that folder.
@BeforeClass	The @BeforeClass annotated method will be executed before the first method of the current class is invoked.
@AfterClass	The @AfterClass annotated method will be invoked after the execution of all the test methods of the current class.
@BeforeMethod	The @BeforeMethod annotated method will be executed before each test method will run.
@AfterMethod	The @AfterMethod annotated method will run after the execution of each test method.
@BeforeGroups	The @BeforeGroups annotated method run only once for a group before the execution of all test cases belonging to that group.
@AfterGroups	The @AfterGroups annotated method run only once for a group after the execution of all test cases belonging to that group.

Figure 3

- @BeforeSuite: The annotated method will be run before all tests in this suite have run.
- @AfterSuite: The annotated method will be run after all tests in this suite have run.
- @BeforeTest: The annotated method will be run before any test method belonging to the classes inside the tag is run.

- @AfterTest: The annotated method will be run after all the test methods belonging to the classes inside the tag have run.
- @BeforeGroups: The list of groups that this configuration method will run before. This method is guaranteed to run shortly before the first test method that belongs to any of these groups is invoked.
- @AfterGroups: The list of groups that this configuration method will run after. This method is guaranteed to run shortly after the last test method that belongs to any of these groups is invoked.
- @BeforeClass: The annotated method will be run before the first test method in the current class is invoked.
- @AfterClass: The annotated method will be run after all the test methods in the current class have been run.
- @BeforeMethod: The annotated method will be run before each test method.
- @AfterMethod: The annotated method will be run after each test method.
- @Test: The annotated method is a part of a test case

9 Assignments

NOTE: Handling Captcha Verification

Captcha cannot be bypassed using automation tools like Selenium. The recommended approach is to solve the Captcha manually while the script waits:

- Add a delay using `Thread.sleep(25000);` or more to allow enough time to manually enter the Captcha and let the page load correctly (manual verification).
- Manually complete the verification process via email or mobile app as required.
- Avoid automating Captcha as it is specifically designed to prevent bots from accessing the page.

Q1: What is the output of this sample code? Provide a detailed description on your answer.

```
import org.testng.annotations.Test;
public class TestNG_Priority_Annotations {
        @Test()
        public void c_method(){
                System.out.println("I'm in method C"); }
        @Test()
        public void b_method(){
                System.out.println("I'm in method B"); }
        @Test(priority=6)
        public void a_method(){
                System.out.println("I'm in method A"); }
        @Test(priority=0)
        public void e_method(){
                System.out.println("I'm in method E"); }
        @Test(priority=6)
        public void d_method(){
                System.out.println("I'm in method D"); }
}
```

Q2: Using Selenium, Write a program to browse "http://demo.guru99.com/test/newtours/" website and retrieve the title of the web page, print it in the output, and test (you can use assertion functions) it to be equal to the phrase "Welcome: Mercury Tours" (hint: you can use `getTitle()` function for your web driver).

Q3: Using Selenium, Write a program to browse "http://www.facebook.com" website and retrieve the tag name of the element *email* text field on the web page and print it in the output. (hint: you can use `getTagName()` function for your web driver).

Q4: Using Selenium and TestNG, Write a program (Java class) to do the following:
Go to the website "https://lambdatest.github.io/sample-todo-app/", find the elements "Second Item" and "Fourth Item" by your driver and check them. Then, find the element of the blank field at the end, clear its content, and add your own name, and submit it (hint: you can use `By.name(...)` and `By.id(...)` functions for your driver).
You need to submit your java code with your output.

Q5: For the first part, using Selenium and TestNG, Write a program containing 4 functions with respect to the following priority from 1 to 4 to be tested (`@Test`) and run in this order:

1. `openBrowser()`: Open Browser say Chrome, Firefox, etc.
2. `launchGoogle()`: Launch "http://www.google.com".
3. `peformSeachAndClick1stLink()`: Perform a search using "Facebook" through Google search field (hint: you can use xpath function in finding the appropriate element on the web page).

4. `FaceBookPageTitleVerification()`: Test and Verify the title of Google search page to be equal to the phrase "Facebook - Google Search". Also, you should use sleep function after performing `click()` function on the search button in Google web page and then, retrieve and test the title of the search result page (hint: using `getTitle().contains(...)` function by your driver and appropriate assertion to test it).
5. `driverexit()`: Quit from the driver.

For the next part, you should modify your implemented program from the previous part by using appropriate TestNG annotations instead of using priority to make the aforementioned methods run in the correct order.

Q6: Using Selenium and TestNG, Write a program (Java class) containing 3 functions (`public void`) named `executSessionOne()`, `executSessionTwo()`, and `executSessionThree()` to be tested (using `@Test`) and run in a parallel format (simoultaneously) using *testng.xml* file.
In each function, use an appropriate driver based on your browser and then go to the website "http://demo.guru99.com/V4/" and find the element *UserID* (hint: you can use `By.name`) and fill it with the phrase "Driver 1", "Driver 2", and "Driver 3", respectively.

Q7: Using Selenium and TestNG, Write a program (Java class) to check whether your login to your LinkedIn/Google/Facebook account is successful. To login to a LinkedIn account as an example, you should browse "https://www.linkedin.com/login" and then, find the elements for username, password, and login button by your appropriate driver (for Firefox, Chrome, etc.) and fill them with your own account information (you may need to use *sleep* function after performing `click()` function).
After that, verify and test the URL after login and compare it with the expected (actual) URL like "https://www.linkedin.com/feed/" to be the same to be considered as a valid login (hint: you can use `getCurrentUrl()` function for your driver as well as an appropriate assertion to test the URLs). Print a suitable message whether it is successful or not.
You need to submit your java code with your output, and also, note that you should cover your own account information (username and password) in your submission.

To securely handle your LinkedIn username and password, use `environment variables` instead of hardcoding credentials. To set up environment variables in Eclipse:

1. Open Eclipse and go to `Run → Run Configurations...`.
2. In the left panel, find and select your Java class (e.g., `LinkedInLoginTest`).
3. Click the **Environment** tab.
4. Click **New...** and add:
   - **Name:** LINKEDIN_USERNAME
   - **Value:** your-email@example.com
   - Click **OK**.
5. Click **New...** again and add:
   - **Name:** LINKEDIN_PASSWORD
   - **Value:** your-secure-password
   - Click **OK**.
6. Click **Apply → Run**.

10 Submission and Marking Instructions

10.1 Submissions

Once all required tasks are completed, you should submit your lab assignment. Follow the instructions below for a valid submission:

- After checking the accuracy and completeness of your assignment tasks, you should export and submit the FULL Eclipse project/folder as a ZIP file containing packages and source code files (for implementation and coding tasks/questions).

  - Individual files (e.g. java or class files) will NOT be accepted as a valid submission since your submitted package or Java project should be run completely and successfully by itself.

- You MUST submit a single PDF file containing required description or explanation for each of the assignment tasks/questions separately including any required justification, graphs, diagrams, test requirements/cases, calculation, pictures/snapshots from tools or IDE, test results, and so forth.

- The **submission deadline** for this lab (Lab III) is the corresponding lab session in **Week 9** (the original deadline is Week 8, however, due to the midterm exam, it is extended one week to Week 9).

- The lab demo and questioning-answering will be held during the lab sessions of the corresponding submission weeks.

10.2 Marking Scheme

- This lab (Lab III) constitutes 4% of your entire grade for this course.

- All assignment tasks/questions in each lab have the same grade.

- The grade for each lab is constituted from 50% for the lab submissions, 10% for the lab attendance, and 40% for demo and questioning-answering during the lab session.

- Note that all the labs constitute 25% of your entire grade for this course.