Selenium with TestNG
Lecture 4

How to Rerun your Failed TestCases in TestNG:

Every time tests fail in a suite, TestNG creates a file called **testng-failed.xml** in the output directory. This XML file contains the necessary information to rerun only these methods that failed, allowing you to quickly reproduce the failures without having to run the entirety of your tests.

Unset

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE suite SYSTEM "https://testng.org/testng-1.0.dtd">
<suite name="Failed suite [Default suite]" guice-stage="DEVELOPMENT">
 <test thread-count="5" name="Default test(failed)">
   <classes>
     <class name="UI.LaunchApp">
       <methods>
         <include name="launchApplication"/>
       </methods>
     </class> <!-- UI.LaunchApp -->
   </classes>
 </test> <!-- Default test(failed) -->
</suite> <!-- Failed suite [Default suite] -->
```

Note that testng-failed.xml will contain all the necessary dependent methods so that you are guaranteed to run the methods that failed without any SKIP failures.

Sometimes, you might want TestNG to automatically retry a test whenever it fails. In those situations, you can use a retry analyzer.

When you bind a retry analyzer to a test, TestNG automatically invokes the retry analyzer to determine if TestNG can retry a test case again in an attempt to see if the test that just fails now passes. Here is how you use a retry analyzer:

- Build an implementation of the interface **org.testng.IRetryAnalyzer**
- Bind this implementation to the **@Test annotation for e.g., @Test(retryAnalyzer = LocalRetry.class)**

Following is a sample implementation of the retry analyzer that retries a test for a maximum of three times.

Unset

```
import org.testng.IRetryAnalyzer;
import org.testng.ITestResult;

public class Retry implements IRetryAnalyzer {

  private int retryCount = 0;
  private static final int maxRetryCount = 3;

  @Override
  public boolean retry(ITestResult result) {
    if (retryCount < maxRetryCount) {
      retryCount++;
      return true;
    }
    return false;
  }
}
```

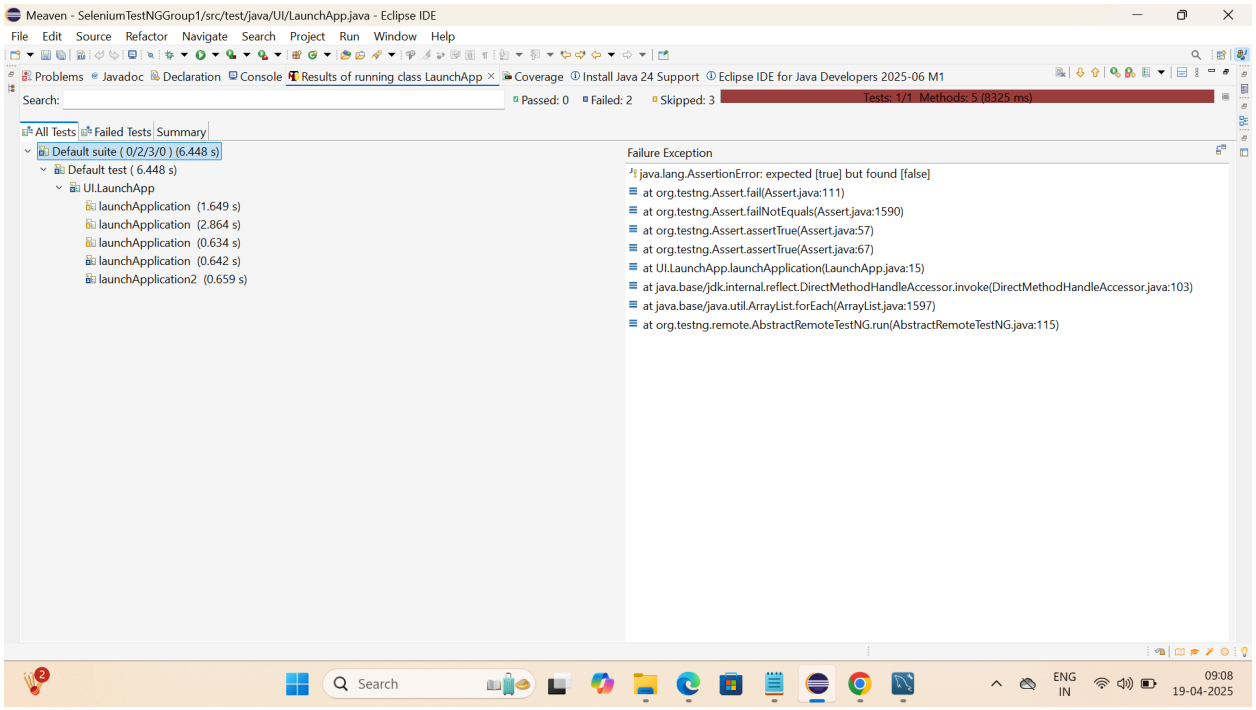
Unset

```
package UI;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.chrome.ChromeDriver;
import org.testng.Assert;
import org.testng.annotations.Test;

public class LaunchApp {
     WebDriver driver = new ChromeDriver();

     @Test(retryAnalyzer = Common.Retry.class)

     public void launchApplication() {
          driver.get("https://www.google.com/");
          Assert.assertTrue(false);
     }
}
```

```
package UI;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.chrome.ChromeDriver;
import org.testng.Assert;
import org.testng.annotations.Test;

public class LaunchApp {
        WebDriver driver = new ChromeDriver();

        @Test(retryAnalyzer = Common.Retry.class)

        public void launchApplication() {
                driver.get("https://www.google.com/");
                Assert.assertTrue(false);
        }

        @Test

        public void launchApplication2() {
```

```
            driver.get("https://www.google.com/");
            Assert.assertTrue(false);
        }

}
```

Here we have two tests and one test is specifying which class to use as retry analyser. If we run this test we will get following results



You can see that Test1 was run 4 times and its been marked failed only at the last run. It was run 4 times because of the retry analyser. However, looking at Test2 we can see that it is run only once. This is obvious because we never specified a **retryAnalyzer** for Test2.

Lets look at the second way of adding retryAnalyzer to your test.

*Specifying **retryAnalyzer** during runtime*

In this case you would need to implement the **ITestAnnotationTransformer** interface. ***ITestAnnotationTransformer*** interface falls under a broad category of interfaces called TestNG Listeners.

```
package Common;
import java.lang.reflect.Constructor;
import java.lang.reflect.Method;
import org.testng.IAnnotationTransformer;
import org.testng.annotations.ITestAnnotation;
public class RetryListener implements IAnnotationTransformer{
	@Override
	public void transform(ITestAnnotation annotation, Class testClass, Constructor testConstructor,
			Method testMethod) {
		annotation.setRetryAnalyzer(Retry.class);

	}
}
```

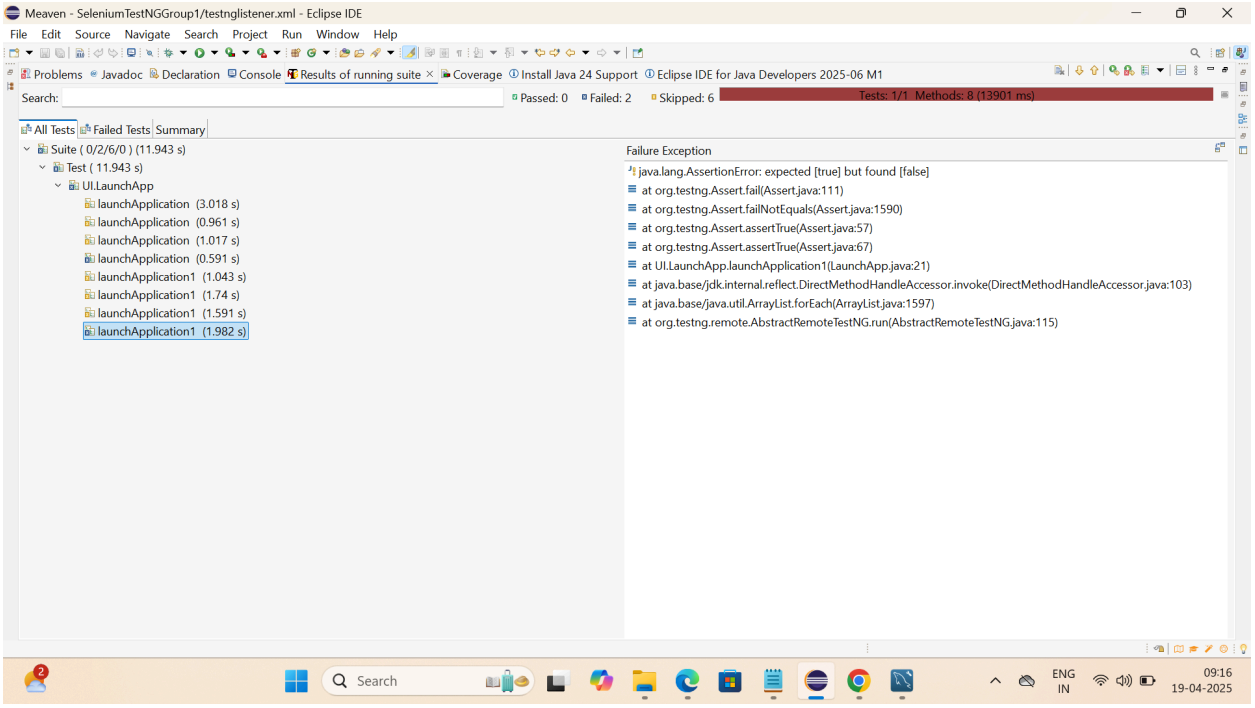
This interface is used to programatically add annotation to your test methods during run time. Transform method is called for every test during test run. A simple implementation of this interface can help us set the retry analyser.

Once we have the implementation of *IAnnotationTransformer*, we just need to add it as a listener in the 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.RetryListener"></listener>
</listeners>
```

```
 <test thread-count="5" name="Test">
   <classes>
     <class name="UI.LaunchApp"/>
   </classes>
 </test> <!-- Test -->
</suite> <!-- Suite -->
```

```
package UI;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.chrome.ChromeDriver;
import org.testng.Assert;
import org.testng.annotations.Test;

public class LaunchApp {
	WebDriver driver = new ChromeDriver();

	@Test

	public void launchApplication() {
		driver.get("https://www.google.com/");
		Assert.assertTrue(false);
	}

	@Test

	public void launchApplication1() {
		driver.get("https://www.google.com/");
		Assert.assertTrue(false);
	}
}
```



You can now simply run the tests and see how both Test1 and Test2 are re-executed 4 times each by TestNG. This is pretty much it on this topic. In the next section we will see how we can use a custom Java Annotation to specify the retry limit count in the test directly.

TestNG Report Generation in Selenium:

Reporter Class is an inbuilt class in TestNG, which is available under the **org.testng** package. This class provides test styles to log dispatches that will be included in the HTML reports generated by TestNG.

Reporter Class is one of the simplest ways of generating log information, where the logs in the reports can be either stoner-generated or system-generated reports.
TestNG Reporting Class is quite useful as it helps analyze failed tests based on the detailed information from the logs. This avoids the need to rerun the entire test case. By

specifying different logs at each step, QAs can use this classification when debugging the failures, making debugging easier.

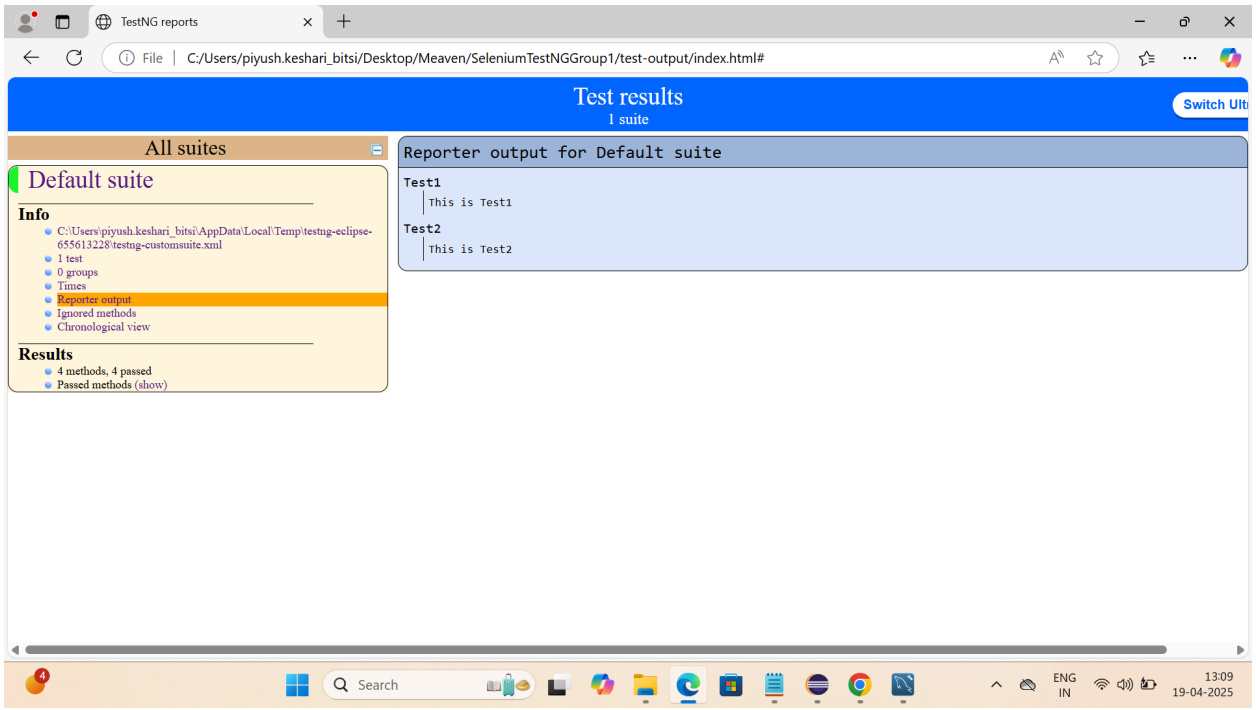
Reporter is a class present in TestNG. It has four different approaches to storing the log information:

- Reporter.log(String s);
- Reporter.log (String s, Boolean logToStandardOut);
- Reporter.log (String s, int level);
- Reporter.log (String s, int level, Boolean logToStandardOut);

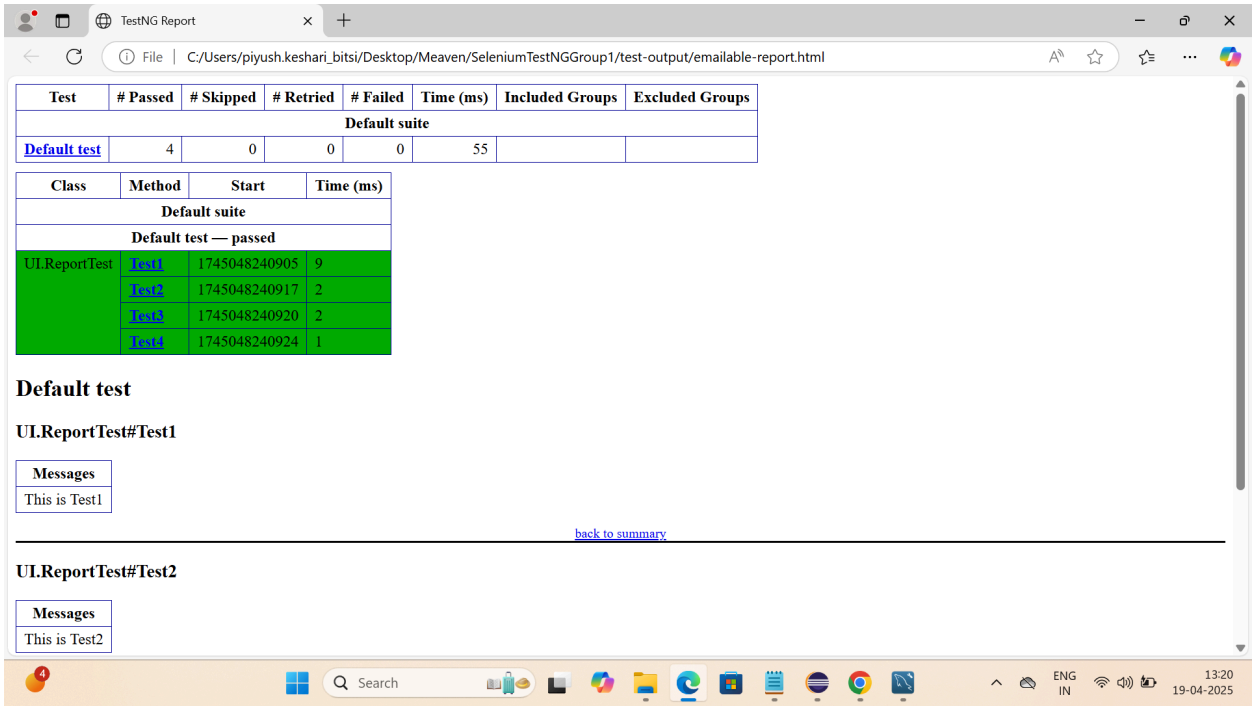
```
package UI;
import org.testng.Reporter;
import org.testng.annotations.Test;

public class ReportTest {

	@Test

	public void Test1() {
		Reporter.log("This is Test1");
		System.out.println("Test1");

	}

	@Test

	public void Test2() {
		Reporter.log("This is Test2");
		System.out.println("Test2");

	}
	@Test

	public void Test3() {
		System.out.println("Test3");

	}
	@Test

	public void Test4() {
```

```
            System.out.println("Test4");

        }
}
```

index.html



Emailable Report



If there are 500 test methods so we can use listeners

ReportTest.java

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

public class ReportTest {

	@Test

	public void Test1() {
		System.out.println("Test1");

	}

	@Test

	public void Test2() {
		System.out.println("Test2");

	}
```

```
	@Test

	public void Test3() {
		System.out.println("Test3");

	}
	@Test

	public void Test4() {
		System.out.println("Test4");

	}
}
```

Listeners.java

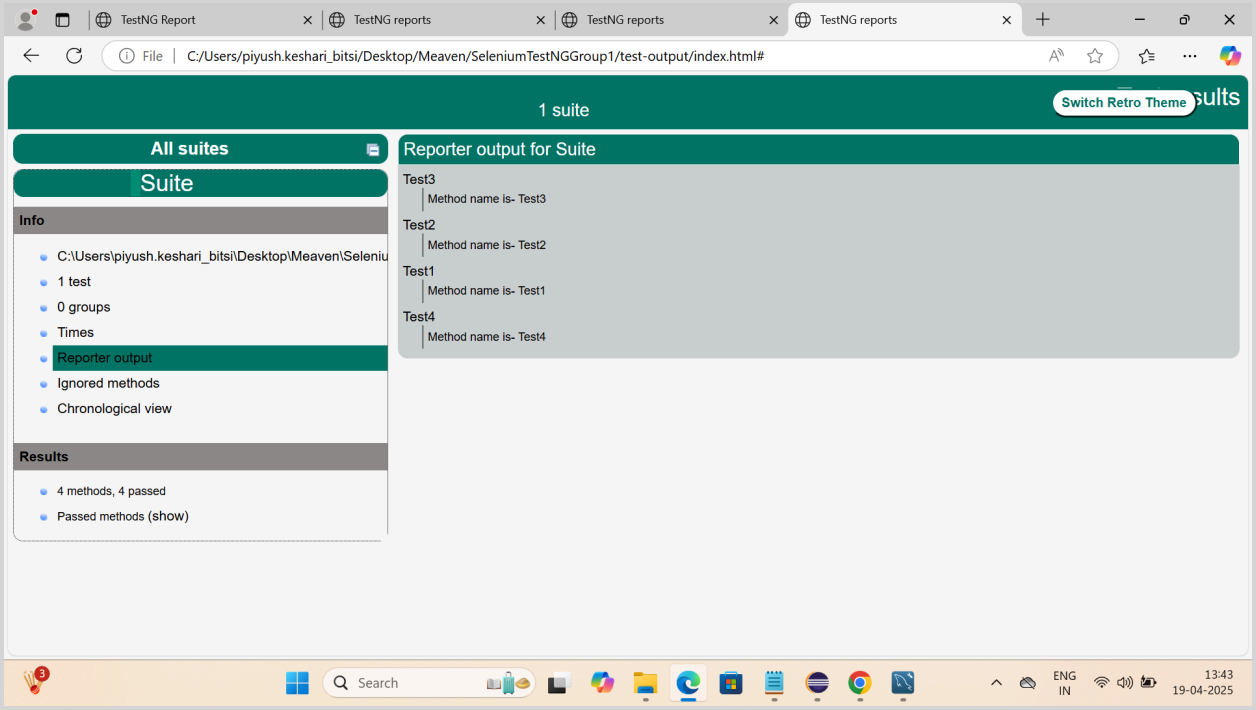
```
package Common;

import java.io.File;
import java.util.Date;

import org.openqa.selenium.OutputType;
import org.openqa.selenium.TakesScreenshot;
import org.testng.ITestContext;
import org.testng.ITestListener;
import org.testng.ITestResult;
import org.testng.Reporter;

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) {
                Reporter.log("Method name is- "+ result.getName());
        System.out.println("Test Starting");

        }

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

        }

        }
```

testngReport.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE suite SYSTEM "https://testng.org/testng-1.0.dtd">
<suite name="Suite">
<listeners>
```

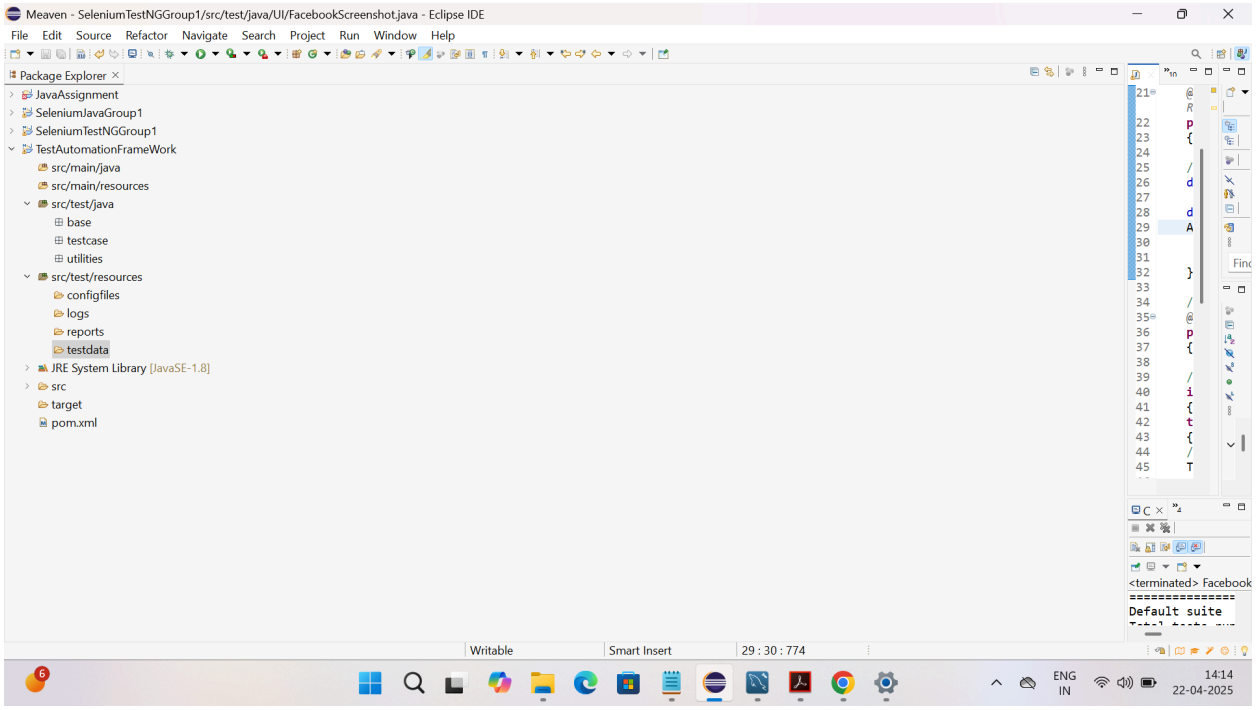
```
<listener class-name="Common.Listeners"></listener>
</listeners>
 <test name="Reports Testing">
   <classes>
     <class name="UI.ReportTest"/>
   </classes>
 </test> <!-- Test -->
</suite> <!-- Suite -->
```



Selenium Framework: Create Project Structure and Understand Basics

Project Structure	Dependencies
Base	Selenium WebDriver
Utilities	TestNG
TestCase	ReportNG
ConfigFiles	Logs-log4j
TestData	Apache POI APIs

Logs	WebDriver Manager
Reports	




```
package testcase;
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.chrome.ChromeDriver;
import io.github.bonigarcia.wdm.WebDriverManager;
public class MyFirstTest {
	public static void main(String[] args) throws InterruptedException {
		WebDriverManager.chromedriver().setup();
		WebDriver driver = new ChromeDriver();
		driver.get("https://www.zoho.com/");
	   driver.manage().window().maximize();
		driver.findElement(By.linkText("Sign In")).click();
		Thread.sleep(2000);
		driver.findElement(By.id("login_id")).sendKeys("piyush.keshari1208@gmail.com");
		Thread.sleep(2000);
		driver.findElement(By.xpath("//button[@id='nextbtn']")).click();
		Thread.sleep(2000);
		driver.findElement(By.xpath("//input[@id='password']")).sendKeys("Piyush@12081990");
		Thread.sleep(2000);
		driver.findElement(By.xpath("//button[@id='nextbtn']")).click();
	}
```

```
}
```