# Selenium Notes

## Initiating driver in selenium

System.*setProperty*("webdriver.chrome.driver","F:\\Drivers\\latestchromedriver\\chromedriver.exe");

WebDriver wd = **new** ChromeDriver();

wd.get("https://www.google.com");

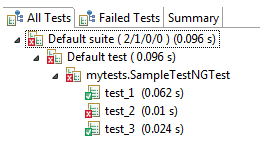
System.***out***.println("browser opened");

# testNG

Default Selenium tests do not generate a proper format for the test results. Using TestNG we can generate test results.

Most Selenium users use this more than Junit because of its advantages. There are so many features of TestNG, but we will only focus on the most important ones that we can use in Selenium. Following are key features of 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, and 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 testing framework can be easily integrated with tools like Maven, Jenkins, etc.
* Annotations used in the testing are very easy to understand ex: @BeforeMethod, @AfterMethod, @BeforeTest, @AfterTest.
* Uncaught exceptions are automatically handled by TestNG without terminating the test prematurely. These exceptions are reported as failed steps in the report.
* WebDriver has no native mechanism for generating reports. TestNG can generate the report in a readable format like the one shown below.



**Advantages of TestNG over JUnit**

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

## Installing testNG for selenium Framework

1. Install TestNG in Eclipse either navigating to "Help -> Install New Software" or from "Eclipse Market Place"
2. Check whether TestNG successfully got installed or not: -In Eclipse Browse to: "Window -> Show View -> Other -> Java" -TestNG must be listed.
3. Add TestNG library from "Project -> Properties -> Java Build Path -> Add Library -> Select TestNG -> Apply -> OK"
4. Create a simple Test Script as follows;

## Sample testNG xml file

### To run tests normal mode

<suite name=*"TestSUITE"*>---name can be any

<test name=*"Sample TEST"*>---name can be any

<classes>

<class name=*"test.extestng"* />----packagename.classname

</classes>

</test>

</suite>

### To run tests in parallel

<suite name=*"TestSUITE"* thread-count=”2” parallel=”tests”>---thread count will decide how many threads can be run in parallel, parallel takes four values

tests All the test cases inside <test> tag of Testing xml file will run parallel.

classes All the test cases inside a Java class will run parallel

methods All the methods with @Test annotation will execute parallel.

instances Test cases in same instance will execute parallel but two methods of two different instances will run in different thread.

<test name=*"Sample TEST"*>---name can be any

<classes>

<class name=*"test.extestng"* />----packagename.classname

</classes>

</test>

</suite>

### To test same testcase on multiple browsers

<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd">

<suite name="TestSuite" thread-count="2" parallel="tests">

<test name="ChromeTest">

<parameter name="browser" value="Chrome" />----this parameter we need to pass in the test case through annotation

<classes>

<class name="parallelTest.CrossBrowserScript">

</class>

</classes>

</test>

<test name="FirefoxTest">

<parameter name="browser" value="Firefox" />

<classes>

<class name="parallelTest.CrossBrowserScript">

</class>

</classes>

</test>

<test name="EdgeTest">

<parameter name="browser" value="Edge" />

<classes>

<class name="parallelTest.CrossBrowserScript">

</class>

</classes>

</test>

</suite>

#### Test class looks like the below

public class CrossBrowserScript {

WebDriver driver;

/\*\*

\* This function will execute before each Test tag in testng.xml

\* @param browser

\* @throws Exception

\*/

@BeforeTest

@Parameters("browser") ----this the annotation that we need to use

public void setup(String browser) throws Exception{

if(browser.equalsIgnoreCase("firefox")){

System.setProperty("webdriver.firefox.marionette", ".\\geckodriver.exe");

driver = new FirefoxDriver();

}

else if(browser.equalsIgnoreCase("chrome")){

System.setProperty("webdriver.chrome.driver",".\\chromedriver.exe");

driver = new ChromeDriver();

}

else{

throw new Exception("Browser is not correct");

}

driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);

}

@Test

public void testParameterWithXML() throws InterruptedException{

//working code

}}

## How to run only failed cases in testNG

In the project workspace you can find testng-failed.xml under test-output folder, it includes the test cases that got failed in the current run, you can run that instead of running the testng.xml file that contains all the test cases.

## TestNG Report Generation in Selenium WebDriver

There are three methods to view report through testNG

1. Under test-ouput folder select emailable-report using web browser
2. Under test-output folder select index.html using web browser
3. Reporter Class

Along with these report generated methods, you can use object.properties file to store the system generated logs as well as user generated logs. But one of the simplest ways to store log information in testing is using Reporter

Use the below commands in the code and we can view this logs in emailable-report and also in index.html

Reporter.log(String s);

Reporter.log(String s, Boolean logToStandardOut);

Reporter.log(String s, int level);

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

## Multiple Test Cases

We can use multiple @Test annotations in a single TestNG file. By default, methods annotated by @Test are executed alphabetically. See the code below. Though the methods c\_test, a\_test, and b\_test are not arranged alphabetically in the code, they will be executed as such.

If you give

@Test(priority=0) for a test then it will be executed before any other tests. In this way we can give numbers as priority to the tests so that it executes from lower value to larger.

## Summary of TestNG Annotations

@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

## Generating Dynamic TestNG XML

package groupid.artifactid;

import java.io.File;

import java.util.ArrayList;

import java.util.List;

import javax.swing.text.Document;

import javax.xml.bind.Element;

import javax.xml.parsers.DocumentBuilder;

import javax.xml.parsers.DocumentBuilderFactory;

import javax.xml.parsers.ParserConfigurationException;

import javax.xml.transform.Transformer;

import javax.xml.transform.TransformerException;

import javax.xml.transform.TransformerFactory;

import javax.xml.transform.dom.DOMSource;

import javax.xml.transform.stream.StreamResult;

import org.testng.TestNG;

import org.testng.annotations.AfterTest;

import org.testng.annotations.BeforeTest;

import org.testng.annotations.Test;

import org.testng.collections.Lists;

import org.testng.reporters.XMLSuiteResultWriter;

import org.testng.xml.XmlSuite;

import org.w3c.dom.Attr;

public class initiateprogram {

public static final String xmlFilePath = "C:\\Users\\ka8678\\workspace\\artifactid\\runtimeTestng.xml";

                @BeforeTest

                public void test()

                {try

                {List<String> testlist = new ArrayList<>();

                                                testlist.add("fftest");

                                                testlist.add("chrometest");

                                                testlist.add("ietest");

                                                DocumentBuilderFactory documentFactory = DocumentBuilderFactory.newInstance();

                                                 DocumentBuilder documentBuilder = documentFactory.newDocumentBuilder();

                                                 org.w3c.dom.Document document = documentBuilder.newDocument();

                                                 // root element

            org.w3c.dom.Element root = document.createElement("suite");

            document.appendChild(root);

            Attr suitename = document.createAttribute("name");

            suitename.setValue("testngTest");

            root.setAttributeNode(suitename);

            org.w3c.dom.Element test = document.createElement("test");

            root.appendChild(test);

            Attr testname = document.createAttribute("name");

            testname.setValue("softwaretestingmaterial");

            test.setAttributeNode(testname);

            org.w3c.dom.Element classes = document.createElement("classes");

            test.appendChild(classes);

             org.w3c.dom.Element class1 = document.createElement("class");

            classes.appendChild(class1);

             Attr classname = document.createAttribute("name");

            classname.setValue("groupid.artifactid.testjava");

            class1.setAttributeNode(classname);

            org.w3c.dom.Element methods = document.createElement("methods");

            class1.appendChild(methods);

            for(int i=0;i<testlist.size();i++)

            {org.w3c.dom.Element include = document.createElement("include");

                methods.appendChild(include);

               Attr incname= document.createAttribute("name");

               incname.setValue(testlist.get(i));

               include.setAttributeNode(incname);}

            // create the xml file and transform the DOM Object to an XML File

            TransformerFactory transformerFactory = TransformerFactory.newInstance();

            Transformer transformer = transformerFactory.newTransformer();

            DOMSource domSource = new DOMSource(document);

            StreamResult streamResult = new StreamResult(new File(xmlFilePath));

            // If you use StreamResult result = new StreamResult(System.out);the output will be pushed to the standard output ...You can use that for debugging

            transformer.transform(domSource, streamResult);

          } catch (ParserConfigurationException pce) {

            pce.printStackTrace();

        } catch (TransformerException tfe) {

            tfe.printStackTrace();}}

                @AfterTest

                public void run()

                {System.out.println("in the method to run......................................");

                                List<String> suites = Lists.newArrayList();

                                suites.add("C:\\Users\\ka8678\\workspace\\artifactid\\runtimeTestng.xml");

                                TestNG tng = new TestNG();

                                tng.setTestSuites(suites);

                                tng.run();

                }}

 testng.xml

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<suite name=*"softwaretestingmaterial"*>

 <test name=*"testngTest"*>

 <classes>

 <class name=*"groupid.artifactid.initiateprogram"* />

 </classes>

 </test>

</suite>

runtimeTestng.xml(generated runtime)

<?xml version="1.0" encoding="UTF-8" standalone="no"?><suite name="testngTest"><test name="softwaretestingmaterial"><classes><class name="groupid.artifactid.testjava"><methods><include name="fftest"/><include name="chrometest"/><include name="ietest"/></methods></class></classes></test></suite>

# Extent-Reporting

Extent reporting have 3 major version 2,3 and 4. You can use any of the latest versions in them as per the requirement. Go to the site <https://extentreports.com/community/>, here click on documents and version you need, you can download the required software from here or from any other domain. You can find the notes related to initializing the specific version methods. Follow them to get to know on how to use those methods.

### Sample program

package test;

import java.io.File;

import org.openqa.selenium.TakesScreenshot;

import java.io.FileInputStream;

import java.io.IOException;

import java.text.SimpleDateFormat;

import java.time.ZonedDateTime;

import java.util.Calendar;

import java.util.Date;

import java.util.Properties;

import java.util.logging.Logger;

import org.apache.commons.io.FileUtils;

import org.openqa.selenium.By;

import org.openqa.selenium.OutputType;

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.WebDriverWait;

import org.testng.annotations.AfterTest;

import org.testng.annotations.Test;

import com.relevantcodes.extentreports.ExtentReports;

import com.relevantcodes.extentreports.ExtentTest;

import com.relevantcodes.extentreports.LogStatus;

@Test

public class TestExtentReport {

Properties obj = new Properties();

GenericMethods gm = new GenericMethods();

Logger log = Logger.getLogger("C:\\Users\\VikramReddyKandi\\eclipse-workspace\\test\\src\\test\\logging.logs");

//C:\Users\VikramReddyKandi\eclipse-workspace\test

ExtentReports report = new ExtentReports(System.getProperty("user.dir")+"\\test-output\\extentreport.html", true);

ExtentTest etest;

public void method1() throws IOException

{

System.out.println(System.getProperty("user.dir")+"\\test-output\\extentreport.html");

etest=report.startTest("method1 testing");

FileInputStream fs = new FileInputStream(System.getProperty("user.dir")+"\\src\\test\\objectrepo.properties");

obj.load(fs);

String test;

test = obj.getProperty("url");

log.info("Driver set----------------------------Logger");

etest.log(LogStatus.INFO, "driver is set");

WebDriver wd = new ChromeDriver();

wd.get("https://www.google.com");

wd.manage().window().maximize();

gm.explicitwait(wd,obj.getProperty("field\_search"),20);

wd.findElement(By.xpath("//input[@title='Search']")).sendKeys("moneycontrol");

WebDriverWait wait = new WebDriverWait(wd, 20);

WebElement we = wait.until(ExpectedConditions.elementToBeClickable(By.xpath("//\*[@class='gNO89b' or @name='btnK']")));

we.click();

etest.log(LogStatus.FAIL,etest.addScreenCapture(capture(wd))+ "Test Failed");

}

public static String capture(WebDriver wd) throws IOException {

TakesScreenshot screen = (TakesScreenshot) wd;

File src = screen.getScreenshotAs(OutputType.FILE);

String dest ="C:\\Users\\VikramReddyKandi\\eclipse-workspace\\test\\src\\testscreenshots\\"+getcurrentdateandtime()+".png";

File target = new File(dest);

FileUtils.copyFile(src, target);

return dest;

}

private static long getcurrentdateandtime() {

Date date = new Date();

long timeMilli = date.getTime();

System.out.println("Time in milliseconds using Date class: " + timeMilli);

Calendar calendar = Calendar.getInstance();

long timeMilli2 = calendar.getTimeInMillis();

return ZonedDateTime.now().toInstant().toEpochMilli();

}

@AfterTest

public void closemethod()

{

System.out.println("browser closed");

report.endTest(etest);

report.flush();

}

}

Testng.xml file:

<suite name=*"TestSUITE"*>

<test name=*"Sample TEST"*>

<classes>

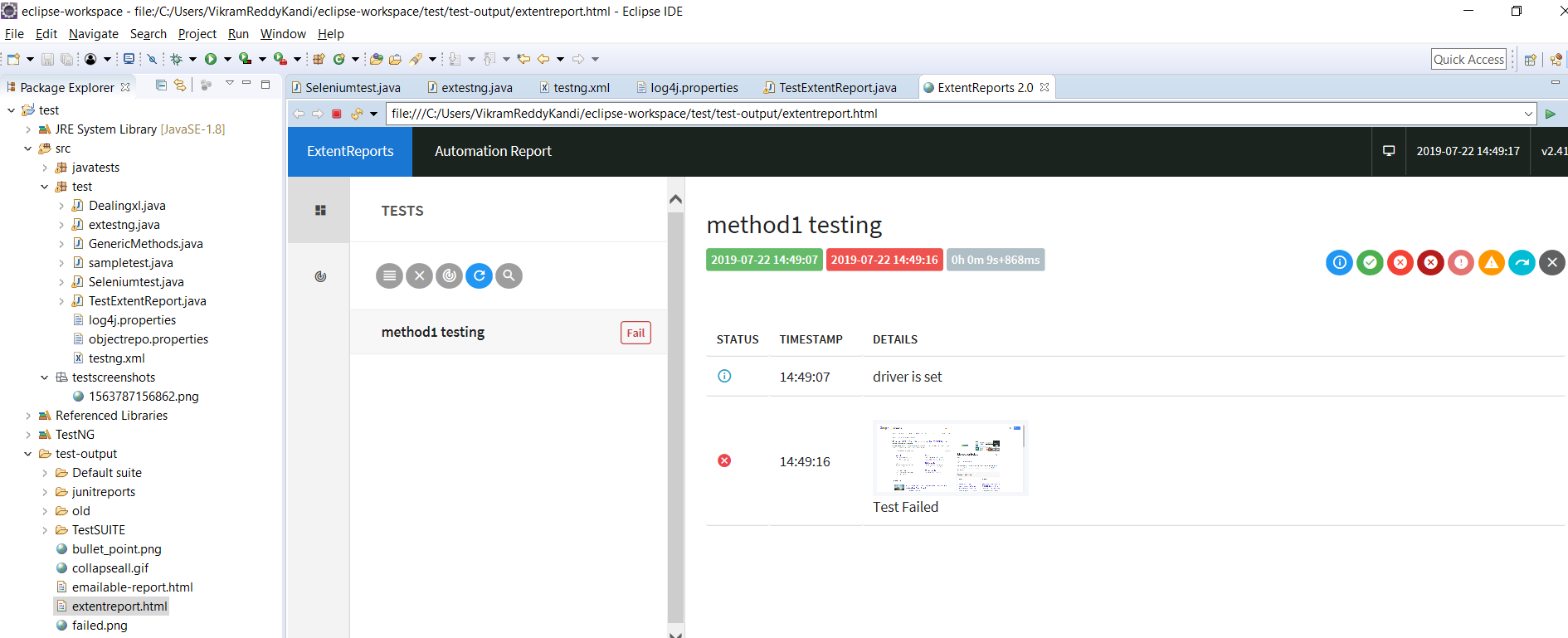
<class name=*"test.TestExtentReport"* />

</classes>

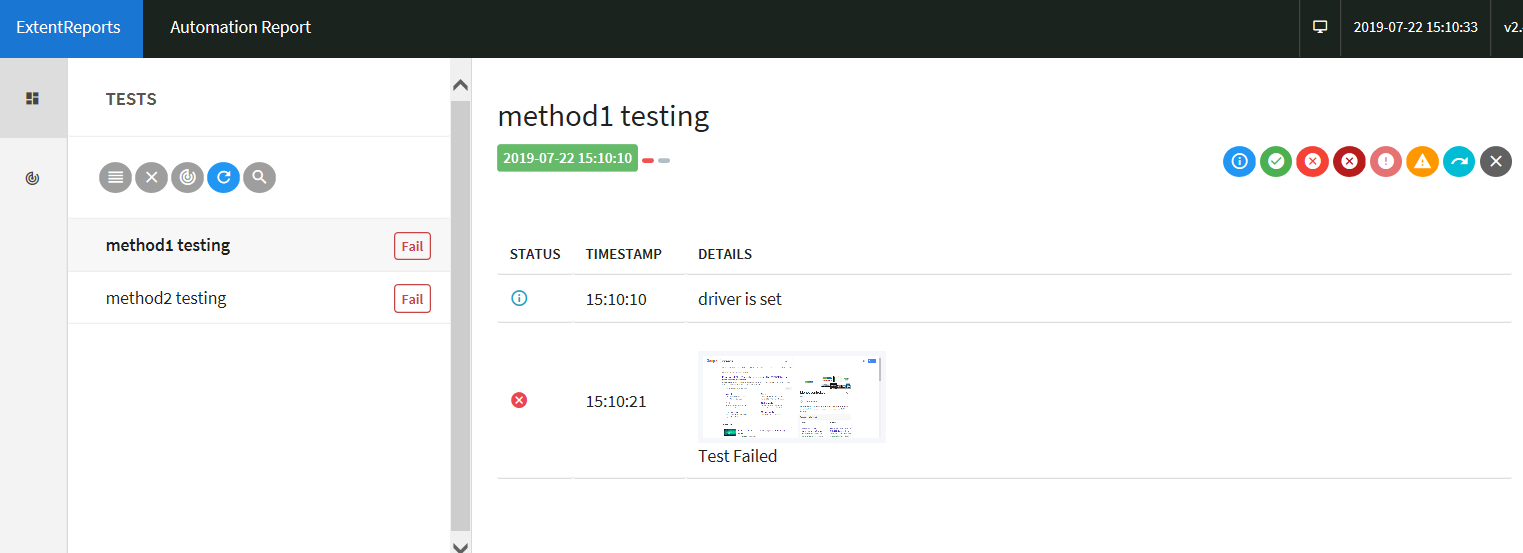
</test>

</suite>

output:



### For multiple tests:



### For parallel tests:

<suite name=*"TestSUITE"* parallel=*"tests"* thread-count=*"2"*>

<test name=*"Sample TEST"*>

<classes>

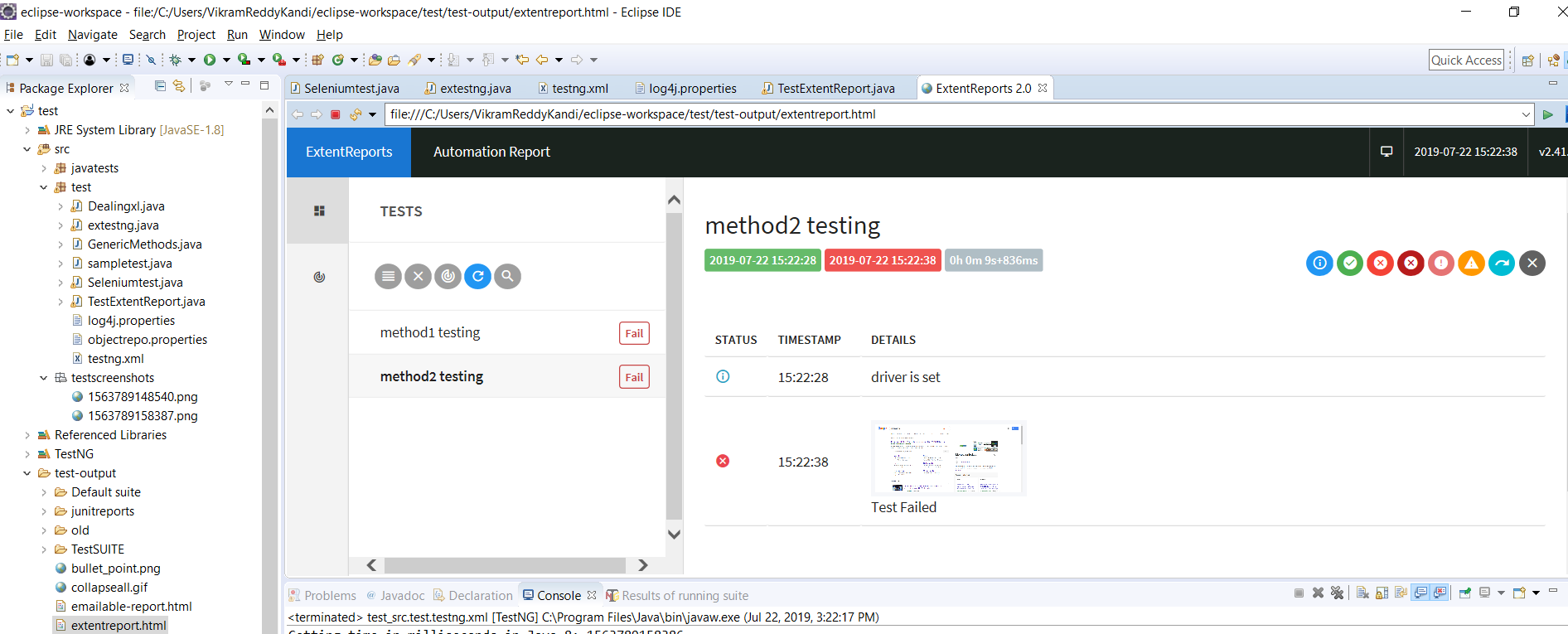
<class name=*"test.TestExtentReport"*/>

</classes>

</test>

</suite>

Running in parallel can be initiated from the tag suite, here you can mention tests, methods or classes as parallel attributes or none for not executing parallel. Thread count mentions who many tests, classes or methods to be run parallel.



# Maven

Maven is build management tool

Where in a project we will be using many jars and for every time we create a project in a new machine then we need to download all the jars.

Or if you share this project to others they need to download all the jars. To overcome this we can use maven.

Where maven is repository in which all the jar providers will register there jar with some mandatory attributes, which we call as dependencies in maven.

In POM.xml we need to add all the dependency information in the respective tags. Maven will take care of downloading them and with this we can remove the conflict of downloading wrong versions, versions will be uniform across all the users.

If you need any specific jars that are confidential to your project or company, then you can create then and include them in maven local repository, where only the employees who have access to that project or company can only access the jars via dependencies.

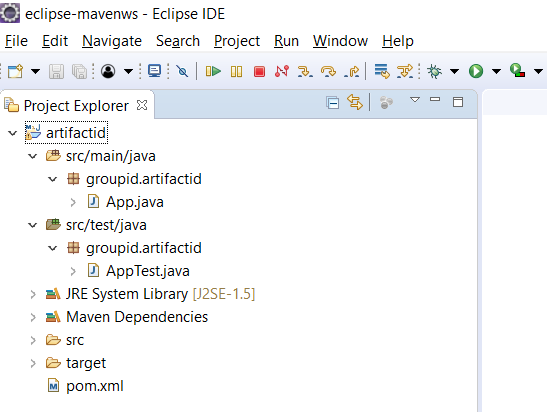
## Setting up Maven project

Install maven from eclipse market place

Type maven in the search box, you will get the latest plugin for maven-eclipse like M2E Settings 2.0.4, select and install it, eclipse asks for restart, accept it.

File—new—maven project

Just provide group id and artifactid and next



Project structure looks like above. Now under test you can create the class that you need.

Before that we need to add dependencies in Pom.xml

Search for maven selenium dependency and copy the stable version, and copy it in pom.xml

Now do the same for maven testng dependency and copy. It looks like below

<dependency>

<groupId>org.seleniumhq.selenium</groupId>

<artifactId>selenium-java</artifactId>

<version>3.14.0</version>

</dependency>

<dependency>

<groupId>org.testng</groupId>

<artifactId>testng</artifactId>

<version>6.8.21</version>

<scope>test</scope>

</dependency>

Additionally, we need to add

maven-compiler-plugin

maven-surefire-plugin

The maven-surefire-plugin is used to configure and execute tests. Here plugin is used to configure the testing.xml for TestNG test and generate test reports.

The maven-compiler-plugin is used to help in compiling the code and using the particular JDK version for compilation. Add all dependencies in the following code snippet, to pom.xml in the <plugin> node

Add the plugin under plugin node or as the below dependency

<dependency>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-surefire-plugin</artifactId>

<version>2.22.2</version>

</dependency>

<dependency>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-compiler-plugin</artifactId>

<version>3.8.1</version>

</dependency>

Create a new class under test folder

**import** org.openqa.selenium.firefox.FirefoxDriver;

**import** org.testng.Assert;

**import** org.testng.annotations.AfterTest;

**import** org.testng.annotations.BeforeTest;

**import** org.testng.annotations.Test;

**public** **class** SampleTest {

**private** WebDriver driver;

@BeforeTest

**public** **void** beforeTest() {

System.*setProperty*("webdriver.chrome.driver","F:\\Drivers\\latestchromedriver\\chromedriver.exe");

}

@Test

**public** **void** method11()

{

WebDriver wd = **new** ChromeDriver();

wd.get("https://www.google.com");

System.***out***.println("browser opened");

//maximizing browser

wd.manage().window().maximize();

System.***out***.println("browser maximized");

wd.findElement(By.*xpath*("//input[@title='Search']")).sendKeys("moneycontrol");

}

}

Now right click on the class and select run as—run configurations—

Select base directory as the path till artifactid, under JRE tab select the jre that is installed in the machine, in the goals field under main tab provide the required goals, example give as test save and then run the program. This is the way that we can run with maven.

We can also run as testng test. Create a testng.xml file, right click on the project(artifactid)—new—other—xml file

Provide the below

<suite name=*"TestSUITE"* parallel=*"tests"* thread-count=*"2"*>

<test name=*"Sample TEST"*>

<classes>

<class name=*"groupid.artifactid.SampleTest"*/>

</classes>

</test>

</suite>

Right click on the testng file and run as testng suite

If you want to use extent report then add the dependency and write code for extent reporting in the class.