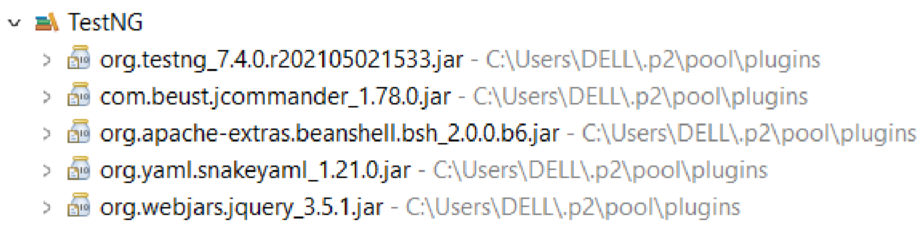
1.Perform All Test Annotations

Step 1.1: Creating a simple Java project

- Open Eclipse.
- Go to the **File** menu and select **New->Java Project.**
- Enter the project name as **Annotations**. Click on **Next.**
- This will create the project files in the Project Explorer.

Step 1.2: Installing TestNG



```
TestNG
  org.testng_7.4.0.r202105021533.jar - C:\Users\DELL\.p2\pool\plugins
  com.beust.jcommander_1.78.0.jar - C:\Users\DELL\.p2\pool\plugins
  org.apache-extras.beanshell.bsh_2.0.0.b6.jar - C:\Users\DELL\.p2\pool\plugins
  org.yaml.snakeyaml_1.21.0.jar - C:\Users\DELL\.p2\pool\plugins
  org.webjars.jquery_3.5.1.jar - C:\Users\DELL\.p2\pool\plugins
```

Step 1.3: Adding TestNG libraries to the Class Path

- In the **Project Explorer**, right-click on **Annotations.**
- Select **Properties**. Select **Java Build Path** from the list. Go to **Libraries.**
- Click on **Add Library.** Select **TestNG**. Click on **Next**. Now, click on **Finish.**
- Finally, click on **Apply and Close.**

Step 1.4: Creating a class file named TestAnnotations

- In the Project Explorer, expand **Annotations->Java Resources.**
- Right-click on **src** and choose **New->Class.**
- In **Class Name,** enter **TestAnnotations**. In **Package Name,** enter **com.testannotations** and click on **Finish.**
- Enter the following code:

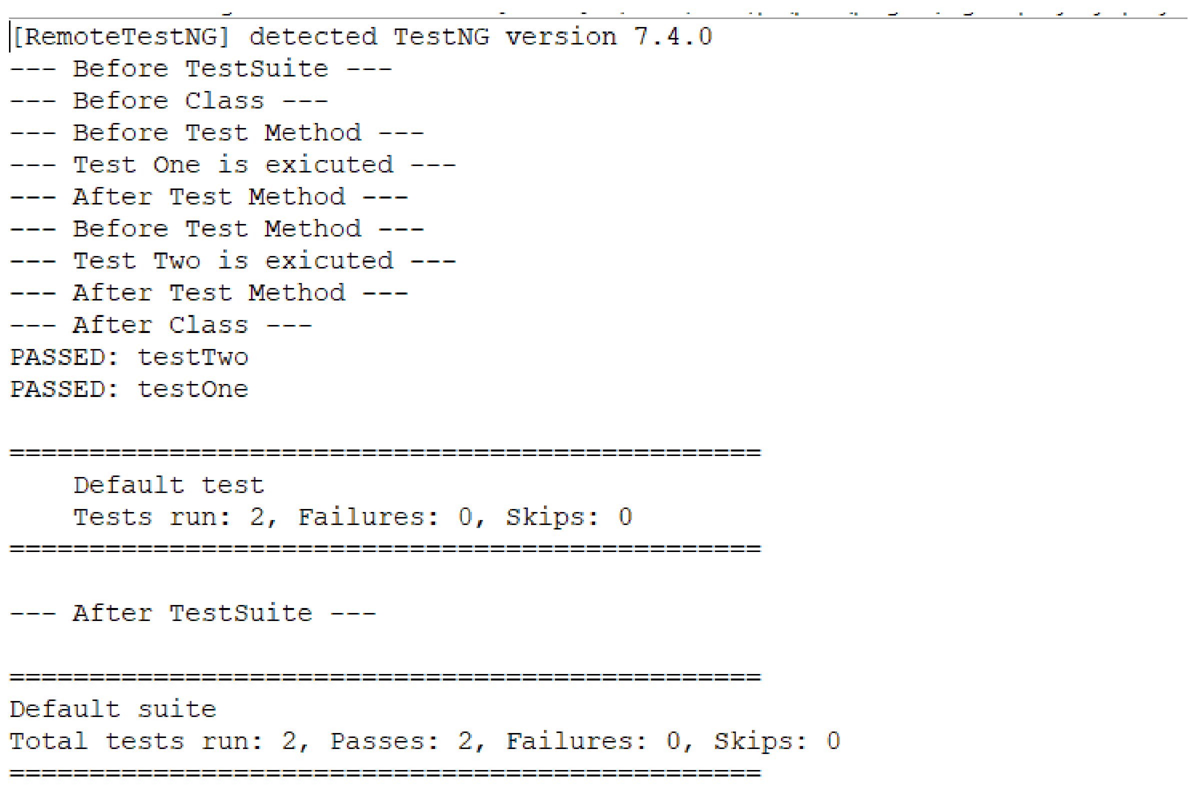
```
package com.simplilearn.testng.phase2_selenium_testng_14_12_23;

import org.testng.annotations.AfterClass;
import org.testng.annotations.AfterMethod;

import org.testng.annotations.AfterSuite;

import org.testng.annotations.BeforeClass;

import org.testng.annotations.BeforeMethod;

import org.testng.annotations.BeforeSuite;

import org.testng.annotations.Ignore;

import org.testng.annotations.Test;



/**

 * This class demonstrate standard annotations available in TestNg
Framework.

 */

public class TestNgStandardAnnotationTest {



      @BeforeSuite

      public void setupBeforeTestSuite() {

            System.out.println("--- Before TestSuite ---");

      }



      @AfterSuite

      public void cleanAfterTestSuite() {

            System.out.println("--- After TestSuite ---");

      }



      @BeforeClass

      public void setupBeforeClass() {

            System.out.println("--- Before Class ---");

      }
```

```
@AfterClass

public void cleanAfterClass() {

      System.out.println("--- After Class ---");

}


@BeforeMethod

public void setupBeforeTestMethod() {

      System.out.println("--- Before Test Method ---");

}


@AfterMethod

public void cleanAfterTestMethod() {

      System.out.println("--- After Test Method ---");

}


@Test

public void testOne() {

      System.out.println("--- Test One is exicuted ---");

}


@Test

public void testTwo() {

      System.out.println("--- Test Two is exicuted ---");

}


@Test

@Ignore
```

```
            public void testThree() {

                  System.out.println("--- Test Three is exicuted ---");

            }



      }
```

Step 1.5: Running the project as TestNG

- Right-click on **TestAnnotations** class. Click on **TestNG->Convert to TestNG.**
- Click on **Finish.** It will create a **TestNG.xml** file. Open that file.
- Right click on the screen. Select **Run As ->TestNG Suite.**



```
[RemoteTestNG] detected TestNG version 7.4.0
--- Before TestSuite ---
--- Before Class ---
--- Before Test Method ---
--- Test One is exicuted ---
--- After Test Method ---
--- Before Test Method ---
--- Test Two is exicuted ---
--- After Test Method ---
--- After Class ---
PASSED: testTwo
PASSED: testOne

===============================================
    Default test
    Tests run: 2, Failures: 0, Skips: 0
===============================================

--- After TestSuite ---

===============================================
Default suite
Total tests run: 2, Passes: 2, Failures: 0, Skips: 0
===============================================
```

2. Group Test Cases and Parallel Test Execution

Step 2.1: Creating a simple Java project

- Open Eclipse.
- Go to the **File** menu and select **New->Java Project.**
- Enter the project name as **Parallel Tests**. Click on **Next.**
- This will create the project files in the Project Explorer.

Step 2.2: Download Selenium WebDriver jar, chromdriver.exe, and firefoxdriver.exe

Step 2.3: Adding the Web Driver dependency in the project

- In the Project Explorer, right-click on **Parallel Tests.**
- Select **Properties**. Select **Java Build Path** from the list. Go to **Libraries**.
- Click on **Add External JARs** and browse the location where you have downloaded the JAR files.
- Select JARs from the **root** folder and the **libs** folder.
- Click on **Apply and Close.**
- Copy the **chromedriver.exe** and **geckodriver.exe**, and paste it your project creating a resource folder.

Step 2.4: Installing TestNG

Step 2.5: Adding TestNG libraries to the Class Path

- In the Project Explorer, right-click on **Parallel Tests.**
- Select **Properties**. Select **Java Build Path** from the list. Go to **Libraries.**
- Click on **Add Library.** Select **TestNG**. Click on **Next**. Now, click on **Finish.**
- Click on **Apply and Close.**

Step 2.6: Creating a Java class named AttributeTest.java

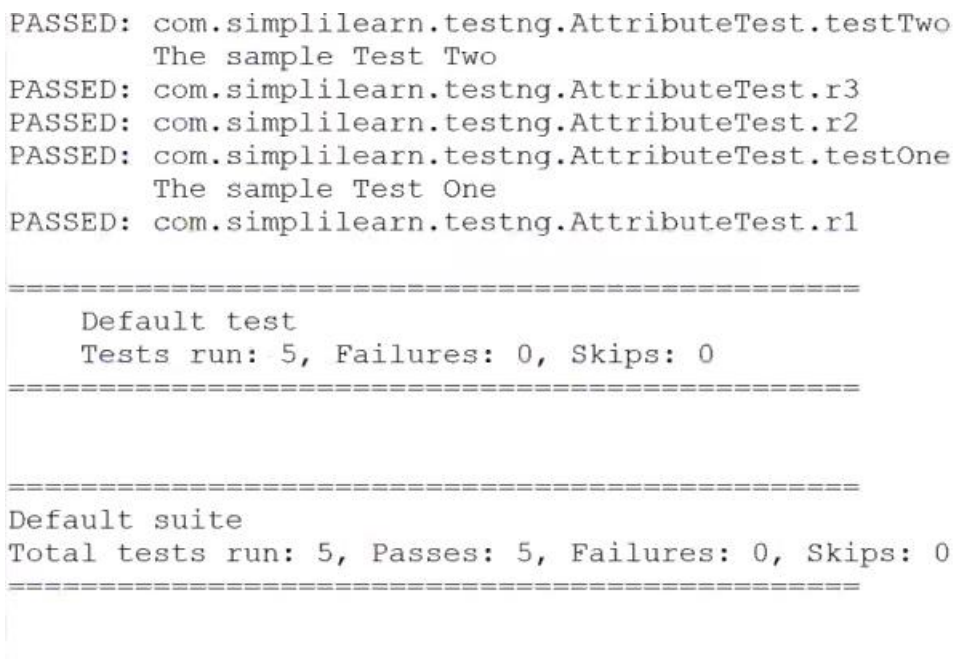
- In the Project Explorer, expand **Parallel Tests->Java Resources.**

- Right-click on **src** and select **New->Class.**
- In **Class Name,** enter **AttributeTests** and click on **Finish.**
- Enter the following code:

```
package com.simplilearn.testng;

import org.testng.annotations.Test;

/**
 * This class demonstrate attribute of @Test.
 */
public class AttributeTest {

   // @Test annotation that describes the information about the test.
         @Test(description = "The sample Test One", priority = 2)
         public void testOne() {
               System.out.println("--- Test One is executed ---");
               // fail();
         }

         // @Test annotation that describes the information about the
test.
         @Test(description = "The sample Test Three", priority = 1 ,
enabled = false)
         public void testThree() {
               System.out.println("--- Test Three is executed ---");
               // fail();
         }

         // @Test annotation, attribute dependOnMethods used
         // When the second test method wants to be dependent on the
first test method.
         @Test(description = "The sample Test Two", dependsOnMethods = {
"testOne", "testThree" }, alwaysRun = true)
         public void testTwo() {
               System.out.println("--- Test Two is executed ---");
         }


         @Test(groups = "Regression Test")
         public void r1() {
               System.out.println("--- Test R1 is executed ---");
         }

         @Test(groups = "Regression Test" , timeOut = 200)
         public void r2() throws InterruptedException {
               // fail();
               Thread.sleep(500);
               System.out.println("--- Test R2 is executed ---");
         }

         @Test(groups = "Regression Test" , dependsOnGroups =
{"Regression Test"})
//       @Test(groups = "Regression Test" , dependsOnGroups =
{"Regression Test"}, alwaysRun = true)
         public void r3() {
               System.out.println("--- Test R3 is executed ---");
         }
   }
```

Step 2.7 Running the project

- Right-click on **AttributeTests** class. Click on **TestNG->Convert to TestNG.**
- Click on **Finish.** It will create a **TestNG.xml** file. Open that file.
- Right click on the screen. Select **Run As ->TestNG Suite.**



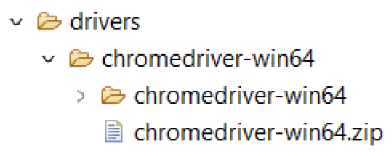
```
PASSED: com.simplilearn.testng.AttributeTest.testTwo
        The sample Test Two
PASSED: com.simplilearn.testng.AttributeTest.r3
PASSED: com.simplilearn.testng.AttributeTest.r2
PASSED: com.simplilearn.testng.AttributeTest.testOne
        The sample Test One
PASSED: com.simplilearn.testng.AttributeTest.r1

===============================================
    Default test
    Tests run: 5, Failures: 0, Skips: 0
===============================================


===============================================
Default suite
Total tests run: 5, Passes: 5, Failures: 0, Skips: 0
===============================================
```

3. Evaluating Test Cases

Step 3.1: Creating a simple Java project

- Open Eclipse.
- Go to the **File** menu. Select **New->Java Project.**
- Enter the project name as **Test Assertions**. Click on **Next.**
- This will create the project files in the Project Explorer.

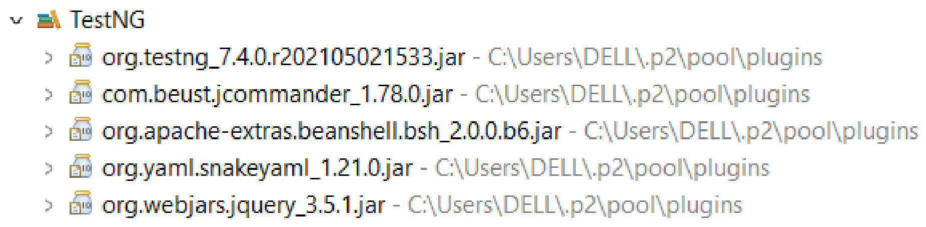
Step 3.2: Downloading Selenium WebDriver jar, chromdriver.exe


drivers
chromedriver-win64
chromedriver-win64
chromedriver-win64.zip

Step 3.3: Adding the WebDriver dependency in the project

- In the Project Explorer, right-click on **Test Assertions.**
- Select **Properties**. Select **Java Build Path** from the list. Go to **Libraries.**
- Click on **Add External JARs** and browse the location where you have downloaded the JAR files.
- Select JARs from the **root** folder and the **libs** folder.
- Click on **Apply and Close.**
- Copy the chromedriver.exe and geckodriver.exe, and paste it to your project creating a resource folder.

Step 3.4: Installing TestNG


TestNG
org.testng_7.4.0.r202105021533.jar - C:\Users\DELL\.p2\pool\plugins
com.beust.jcommander_1.78.0.jar - C:\Users\DELL\.p2\pool\plugins
org.apache-extras.beanshell.bsh_2.0.0.b6.jar - C:\Users\DELL\.p2\pool\plugins
org.yaml.snakeyaml_1.21.0.jar - C:\Users\DELL\.p2\pool\plugins
org.webjars.jquery_3.5.1.jar - C:\Users\DELL\.p2\pool\plugins

Step 3.5: Adding TestNG libraries to the Class Path

- In the Project Explorer, right-click on **AssertionTest**
- Select **Properties**. Select **Java Build Path** from the list. Go to **Libraries.**
- Click on **Add Library.** Select **TestNG**. Click on **Next**. Click on **Finish.**
- Click on **Apply and Close.**

Step 3.6: Creating a Java class named AssertionTest.java

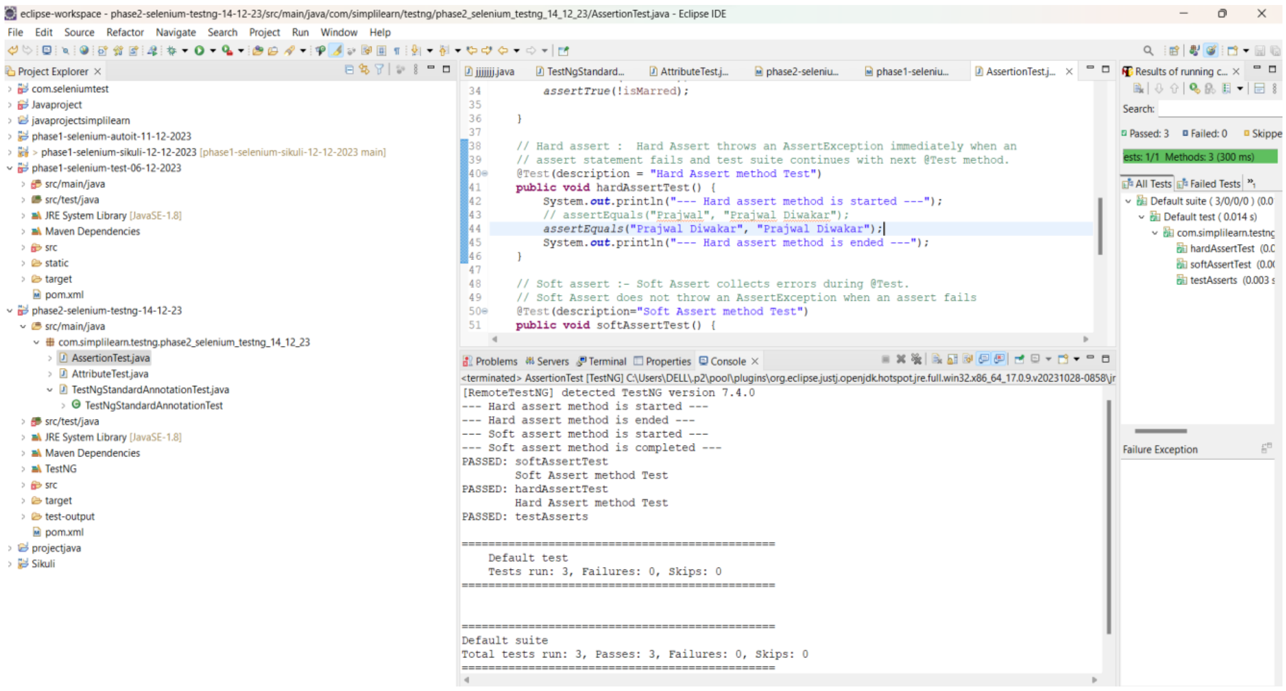
- In the Project Explorer, expand **AssertionTest->Java Resources**
- Right-click on **src** and select **New->Class**
- In **Class Name,** enter **Assertions** and click on **Finish.** In **Package Name,** enter **com.assert** and click on **Finish.**

- Enter the following code:

```
package com.simplilearn.testng.phase2_selenium_testng_14_12_23;

import static org.testng.Assert.assertEquals;
import static org.testng.Assert.assertFalse;
import static org.testng.Assert.assertNotEquals;
import static org.testng.Assert.assertNotNull;
import static org.testng.Assert.assertNull;
import static org.testng.Assert.assertTrue;

import org.testng.annotations.Test;
import org.testng.asserts.SoftAssert;

/**
 * This class demonstrate TestNg Assertion's.
 */
public class AssertionTest {

    @Test
    public void testAsserts() {

            String userName = "Prajwal Diwakar";
            String message = null;
            int age = 30;
            boolean isMarred = false;

            assertEquals("Prajwal Diwakar", userName);
            assertEquals(30, age);
            assertNotEquals("Prajwal", userName);

            assertNotNull(userName);
            assertNull(message);

            assertFalse(isMarred);
            assertTrue(!isMarred);

    }

    // Hard assert :  Hard Assert throws an AssertException immediately 
when an
    // assert statement fails and test suite continues with next @Test 
method.
    @Test(description = "Hard Assert method Test")
    public void hardAssertTest() {
            System.out.println("--- Hard assert method is started ---");
            // assertEquals("Prajwal", "Prajwal Diwakar");
            assertEquals("Prajwal Diwakar", "Prajwal Diwakar");
            System.out.println("--- Hard assert method is ended ---");
    }

    // Soft assert :- Soft Assert collects errors during @Test.
    // Soft Assert does not throw an AssertException when an assert fails
    @Test(description="Soft Assert method Test")
    public void softAssertTest() {
            SoftAssert softAssert = new SoftAssert();
            System.out.println("--- Soft assert method is started ---");
            softAssert.assertEquals("Hello", "Hi");
            softAssert.assertEquals("Prajwal Diwakar", "Prajwal Diwakar");
            softAssert.assertEquals(true, false);
            softAssert.assertTrue(true);
            System.out.println("--- Soft assert method is completed ---");
        }
    }
```

Step 3.7: Running the project

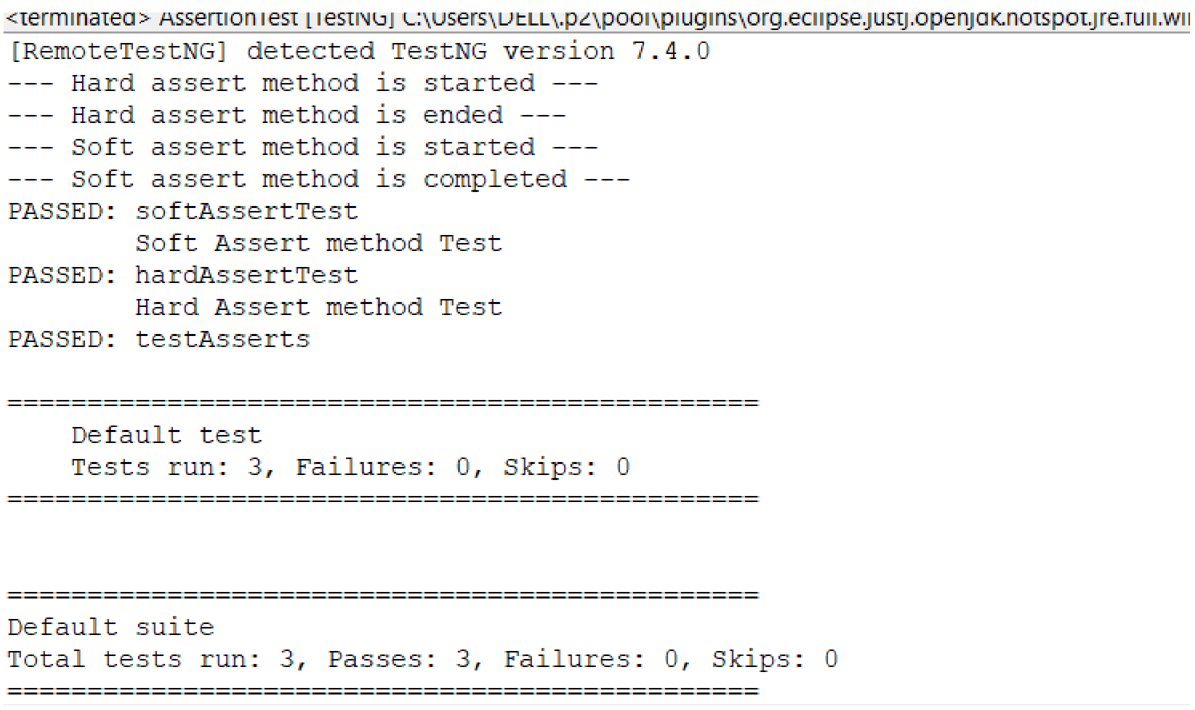
- Right-click on **Assertions** class. Click on **TestNG->Convert to TestNG.**
- Click on **Finish.** It will create a **TestNG.xml** file. Open that file.
- Right-click on the screen. Select **Run As ->TestNG Suite.**



```
        assertTrue(!isMarred);

    }

    // Hard assert :  Hard Assert throws an AssertException immediately when an
    // assert statement fails and test suite continues with next @Test method.
    @Test(description = "Hard Assert method Test")
    public void hardAssertTest() {
        System.out.println("--- Hard assert method is started ---");
        // assertEquals("Prajwal", "Prajwal Diwakar");
        assertEquals("Prajwal Diwakar", "Prajwal Diwakar");
        System.out.println("--- Hard assert method is ended ---");
    }

    // Soft assert :- Soft Assert collects errors during @Test.
    // Soft Assert does not throw an AssertException when an assert fails
    @Test(description="Soft Assert method Test")
    public void softAssertTest() {
```

```
<terminated> AssertionTest [TestNG] C:\Users\DELL\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.9.v20231028-0858\jr
[RemoteTestNG] detected TestNG version 7.4.0
--- Hard assert method is started ---
--- Hard assert method is ended ---
--- Soft assert method is started ---
--- Soft assert method is completed ---
PASSED: softAssertTest
        Soft Assert method Test
PASSED: hardAssertTest
        Hard Assert method Test
PASSED: testAsserts

===============================================
    Default test
    Tests run: 3, Failures: 0, Skips: 0
===============================================


===============================================
Default suite
Total tests run: 3, Passes: 3, Failures: 0, Skips: 0
===============================================
```

eclipse-workspace - phase2-selenium-testng-14-12-23/src/main/java/com/simplilearn/testng/phase2_selenium_testng_14_12_23/AssertionTest.java - Eclipse IDE
Project Explorer
AssertionTest.java
Results of running c...
Passed: 3
Failed: 0
Tests: 1/1 Methods: 3 (300 ms)
Default suite (3/0/0/0)
Default test (0.014 s)
hardAssertTest
softAssertTest
testAsserts
Failure Exception



```
<terminated> AssertionTest [TestNG] C:\Users\DELL\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.wi
[RemoteTestNG] detected TestNG version 7.4.0
--- Hard assert method is started ---
--- Hard assert method is ended ---
--- Soft assert method is started ---
--- Soft assert method is completed ---
PASSED: softAssertTest
        Soft Assert method Test
PASSED: hardAssertTest
        Hard Assert method Test
PASSED: testAsserts

===============================================
    Default test
    Tests run: 3, Failures: 0, Skips: 0
===============================================


===============================================
Default suite
Total tests run: 3, Passes: 3, Failures: 0, Skips: 0
===============================================
```

4.Reports Using Extent Reports

Step 4.1: Generating Extent Reports

- Open Eclipse.

- Create a TestNG project in Eclipse.
- Extent Reports jar file is already present in your practice lab in /home/ubuntu/libs directory. Refer QA to QE lab guide for Phase 2 for more information.
- Add the Extent Reports jar file to your project.
- Create a java class, say **Amazon** and add the following code to it.
- Sample code:

```java
package com.simplilearn.testng.phase2_selenium_testng_14_12_23;

import static org.testng.Assert.assertEquals;

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.firefox.FirefoxDriver;
import org.openqa.selenium.support.ui.ExpectedConditions;
import org.openqa.selenium.support.ui.WebDriverWait;
import org.testng.annotations.AfterMethod;
import org.testng.annotations.BeforeMethod;
import org.testng.annotations.Test;

/**
 * This class demonstrate test for amazon webpage. *
 */
public class AmazonLinkVerificationTest {

   // step1: formulate a test domain url & driver path
         String siteUrl = "https://www.amazon.in/";
         String driverPath = "drivers/chromedriver-win64/chromedriver-
win64/chromedriver.exe";
         WebDriver driver;
         WebDriverWait wait;

         @BeforeMethod
         public void setUp() {

               // step2: set system properties for selenium dirver
               System.setProperty("webdriver.chrome.driver",
driverPath);

               // step3: instantiate selenium webdriver
               driver = new ChromeDriver();

               // step4: launch browser
               driver.get(siteUrl);
         }

         @AfterMethod
         public void cleanUp() {
               driver.quit(); // the quit() method closes all browser
windows and ends the WebDriver session.
               // driver.close(); // the close() closes only the current
window on which
               // Selenium is running automated tests.The WebDriver
session, however, remains
               // active.
         }

         @Test(description = "Test Amazon Mobile Phones Title Match")
         public void linkTest1() throws InterruptedException {
               WebElement link =
driver.findElement(By.xpath("//*[@id=\"nav-xshop\"]/a[4]"));

               link.click();

               // add delay
               Thread.sleep(1000);

               String expectedTitle = "Amazon.in Bestsellers: The most
popular items on Amazon";
               String actualTitle = driver.getTitle();
               assertEquals(actualTitle, expectedTitle);
         }

         @Test(description = "Verify the Todays deals link")
         public void linkTest2() throws InterruptedException {
               WebElement link =
driver.findElement(By.cssSelector("#nav-xshop > a:nth-child(6)"));

               link.click();
```

```
                // add delay
                Thread.sleep(1000);

                String expectedTitle = "Amazon.in - Deals";
                String actualTitle = driver.getTitle();
                assertEquals(actualTitle, expectedTitle);
        }


}
```

Step 4.2: Describing Extent-config.xml

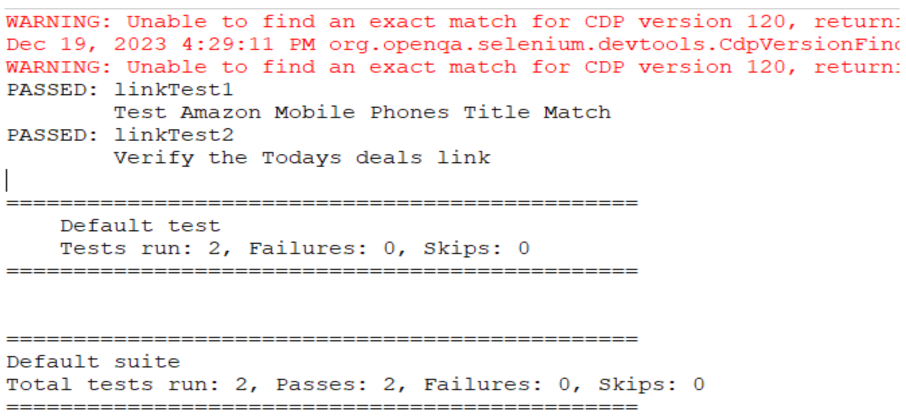
By using this external XML file (extent-config.xml), we could change the details, such as Report Theme, Report Title, and Document Title.

We use the extent object and use loadConfig() method to load this XML file.

```
<?xml version="1.0" encoding="UTF-8"?>
<suite name="eCom Webapp Test Suite">
      <listeners>
            <listener class-name="org.uncommons.reportng.HTMLReporter" />
            <listener class-name="org.uncommons.reportng.JUnitXMLReporter"
/>
      </listeners>
      <test name="Amazon Test">
            <classes>
                  <class
name="com.simplilearn.testng.amazon.AmazonHomePageTest"></class>
                  <class

      name="com.simplilearn.testng.amazon.AmazonLinkVerificationCssSelector
Test"></class>
                  <class

      name="com.simplilearn.testng.amazon.AmazonLinkVerificationTest"></cla
ss>
                  <class

      name="com.simplilearn.testng.amazon.AmazonLinkVerificationXpathTest">
</class>
                  <class
name="com.simplilearn.testng.amazon.AmazonSearchTest"></class>
            </classes>
      </test>
      <test name="Group Test">
            <classes>
```

```
                        <class
name="com.simplilearn.testng.AttributeGroupTest"></class>
                <classes>
        </test>
        <test name="Other Test">
                <classes>
                        <class

        name="com.simplilearn.testng.TestNgStandardAnnotationTest"></class>
                        <class
name="com.simplilearn.testng.AssertionTest"></class>
                </classes>
        </test>
</suite>
```

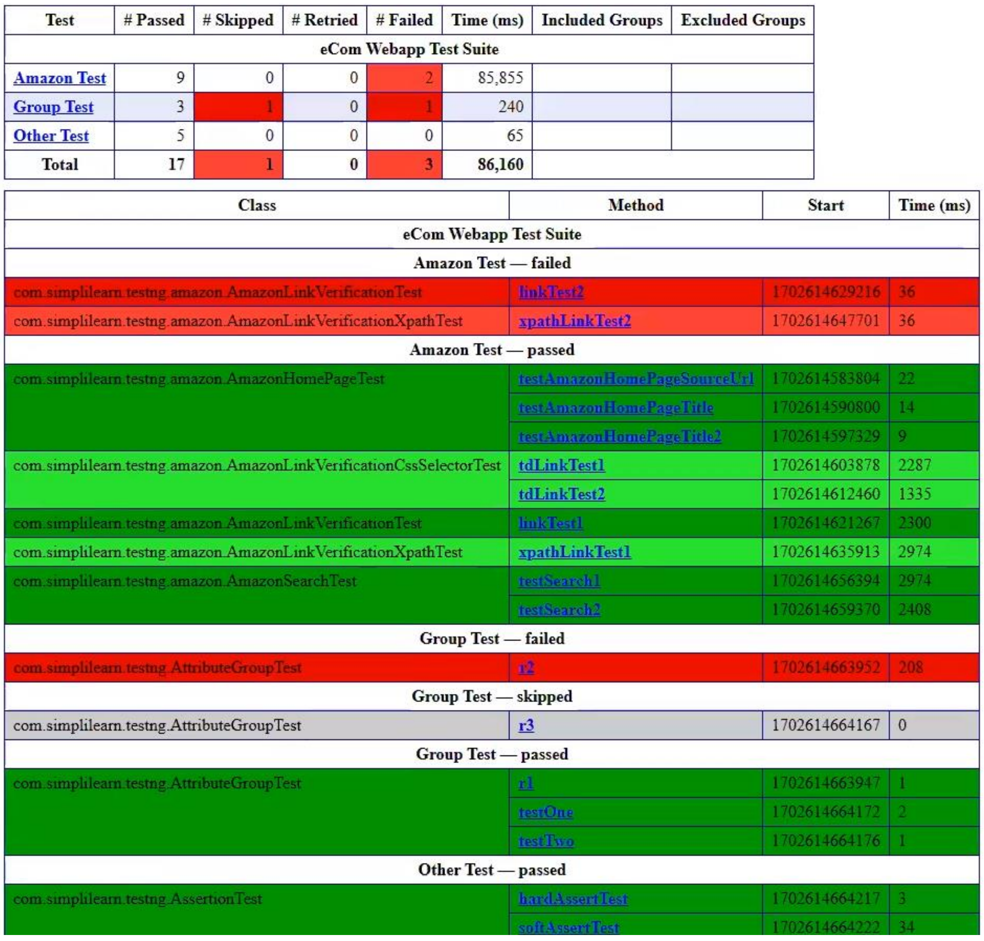
- Console Output:



```
WARNING: Unable to find an exact match for CDP version 120, return:
Dec 19, 2023 4:29:11 PM org.openqa.selenium.devtools.CdpVersionFind
WARNING: Unable to find an exact match for CDP version 120, return:
PASSED: linkTest1
        Test Amazon Mobile Phones Title Match
PASSED: linkTest2
        Verify the Todays deals link
|
===============================================
    Default test
    Tests run: 2, Failures: 0, Skips: 0
===============================================


===============================================
Default suite
Total tests run: 2, Passes: 2, Failures: 0, Skips: 0
===============================================
```

Refresh the project after execution of the above ExtentReportsClass.java file. You can find an HTML file named **STMExtentReport.html** in your test-output folder. Copy the location of the STMExtentReport.html file and open it by using any browser. Once you open the report, you will see rich HTML test results as shown below.

Step 4.3: Fetching Test Summary Report



Test	# Passed	# Skipped	# Retried	# Failed	Time (ms)	Included Groups	Excluded Groups
eCom Webapp Test Suite							
Amazon Test	9	0	0	2	85,855		
Group Test	3	1	0	1	240		
Other Test	5	0	0	0	65		
Total	**17**	**1**	**0**	**3**	**86,160**		

Class	Method	Start	Time (ms)
eCom Webapp Test Suite			
Amazon Test — failed			
com.simplilearn.testng.amazon.AmazonLinkVerificationTest	linkTest2	1702614629216	36
com.simplilearn.testng.amazon.AmazonLinkVerificationXpathTest	xpathLinkTest2	1702614647701	36
Amazon Test — passed			
com.simplilearn.testng.amazon.AmazonHomePageTest	testAmazonHomePageSourceUrl	1702614583804	22
	testAmazonHomePageTitle	1702614590800	14
	testAmazonHomePageTitle2	1702614597329	9
com.simplilearn.testng.amazon.AmazonLinkVerificationCssSelectorTest	tdLinkTest1	1702614603878	2287
	tdLinkTest2	1702614612460	1335
com.simplilearn.testng.amazon.AmazonLinkVerificationTest	linkTest1	1702614621267	2300
com.simplilearn.testng.amazon.AmazonLinkVerificationXpathTest	xpathLinkTest1	1702614635913	2974
com.simplilearn.testng.amazon.AmazonSearchTest	testSearch1	1702614656394	2974
	testSearch2	1702614659370	2408
Group Test — failed			
com.simplilearn.testng.AttributeGroupTest	r2	1702614663952	208
Group Test — skipped			
com.simplilearn.testng.AttributeGroupTest	r3	1702614664167	0
Group Test — passed			
com.simplilearn.testng.AttributeGroupTest	r1	1702614663947	1
	testOne	1702614664172	2
	testTwo	1702614664176	1
Other Test — passed			
com.simplilearn.testng.AssertionTest	hardAssertTest	1702614664217	3
	softAssertTest	1702614664222	34

5. Exporting Reports in Excel

Step 5.1 Creating a project with test cases with multiple annotations

- Open the Eclipse and create a Java project.
- Create multiple test case classes(Say Test_01, Test02).
- Create a Base class to extend the test cases.

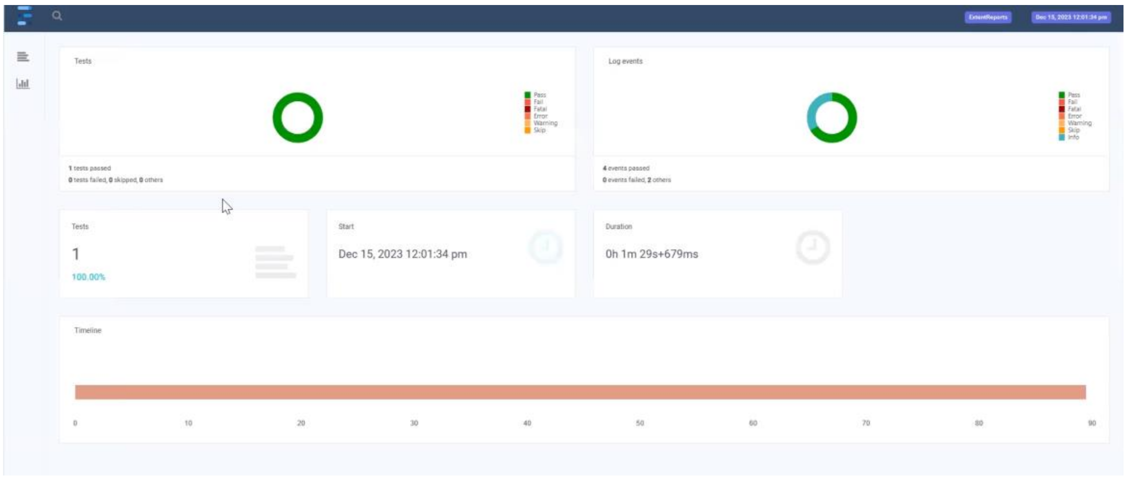
Step 5.2 Adding AT Excel report jars

- Extent Reports jar file is already present in your practice lab in /home/ubuntu/libs directory.
- Add the Extent Reports jar file to your project: Right-click on project->Build path->Configure build path->Add external Jars.
- Click on Apply and then click OK.

Step 5.3 Executing the test suites to see the generated report in Excel sheet

```
package com.simplilearn.testng;

import static org.testng.Assert.assertEquals;

import java.sql.Driver;

import org.openqa.selenium.WebDriver;
import org.openqa.selenium.chrome.ChromeDriver;

import com.aventstack.extentreports.ExtentReports;
import com.aventstack.extentreports.ExtentTest;
import com.aventstack.extentreports.Status;
import com.aventstack.extentreports.reporter.ExtentSparkReporter;

import io.github.bonigarcia.wdm.WebDriverManager;

/**
 * This class demonstrate extent report creation.
 *
 * @author khanw
 *
 */
public class ExtentReport {

// step1: formulate a test domain url & driver path
static String siteUrl = "https://www.amazon.in/";
static String driverPath = "drivers/windows/chromedriver.exe";
static WebDriver driver;


public static void main(String[] args) {
// create the htmlReporter object
ExtentSparkReporter htmlReporter = new ExtentSparkReporter("extentReport.html");


// create ExtentReport and attach this reports
ExtentReports extentReports = new ExtentReports();
extentReports.attachReporter(htmlReporter);


// create a test and add logs

        ExtentTest test1 = extentReports.createTest("Amazon Title Match Test", "Test Amazon
Home Page Title Match");
// initialize and start the browser
WebDriverManager.chromedriver().setup();
// set system properties for selenium dirver
System.setProperty("webdriver.chrome.driver", driverPath);
driver = new ChromeDriver();
test1.log(Status.INFO, "Starting test case");
// maximize the browser window
driver.manage().window().maximize();
test1.pass("
// Navigate Open amazon.in
driver.get(siteUrl);
test1.pass("Navigate Open www.amazon.in");
String expectedTitle = "Online Shopping site in India: Shop Online for Mobiles, Books,
Watches, Shoes and More - Amazon.in";
String actualTitle = driver.getTitle();
assertEquals(actualTitle, expectedTitle);
test1.pass("The title are matched");


driver.quit();

test1.pass("The Browser is quited");

test1.log(Status.INFO, "Completed test case");


extentReports.flush();


}

}
```

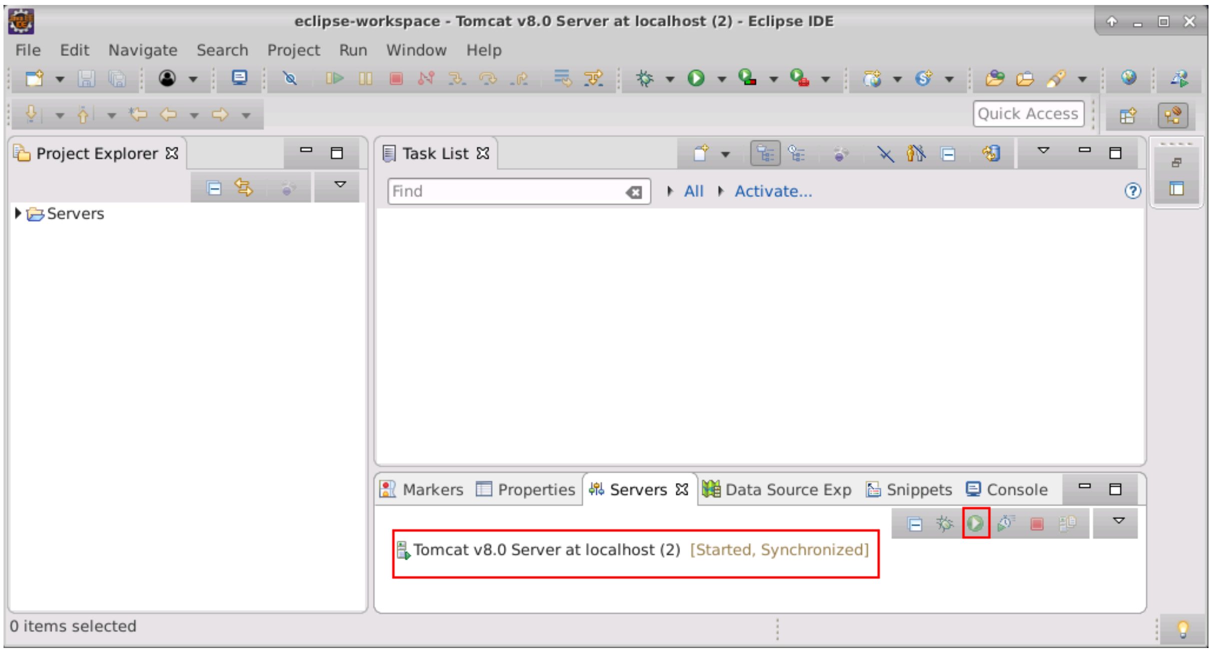
Finally, the executed script can generate the report in Excel and the graph will look like :



6. Publishing Reports on Tomcat

Step 6.1: Starting Tomcat server

- Download Tomcat within Eclipse using the following steps:
  - Open your eclipse environment from the desktop.
  - Select File tab, click on New>Other.
  - In the next page, click on Server from the server folder drop down. Click on Next.
  - Type tomcat in the filter text field. Select the tomcat version that you would like to install and click on Next.
  - Click on Download and Install. Select the directory where you want to download tomcat. Select the installed JRE and click Finish.
- Start the tomcat server in Eclipse


eclipse-workspace - Tomcat v8.0 Server at localhost (2) - Eclipse IDE
File Edit Navigate Search Project Run Window Help
Project Explorer
Servers
Task List
Markers Properties Servers Data Source Exp Snippets Console
0 items selected

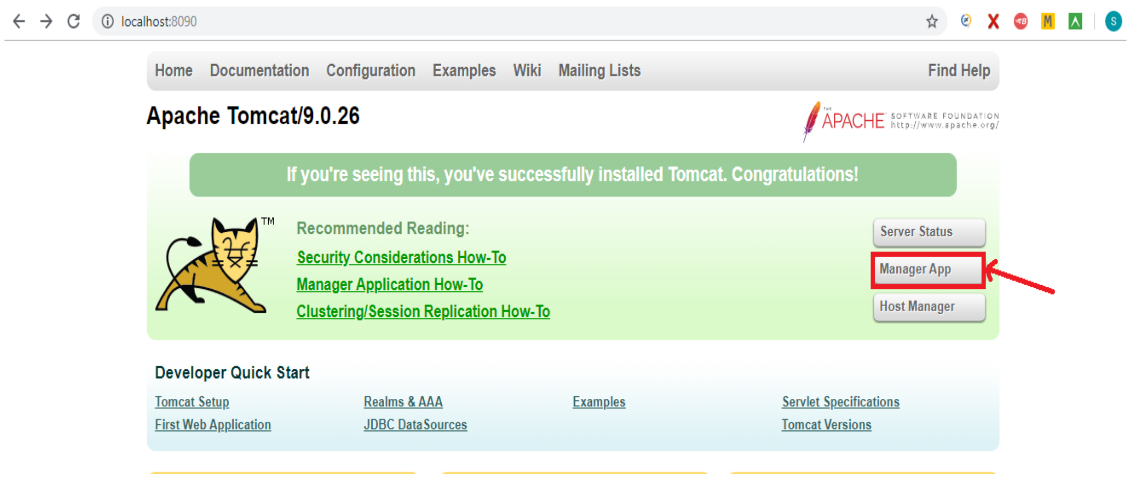
Tomcat v8.0 Server at localhost (2) [Started, Synchronized]

Step 6.2: Creating WAR file from Eclipse.

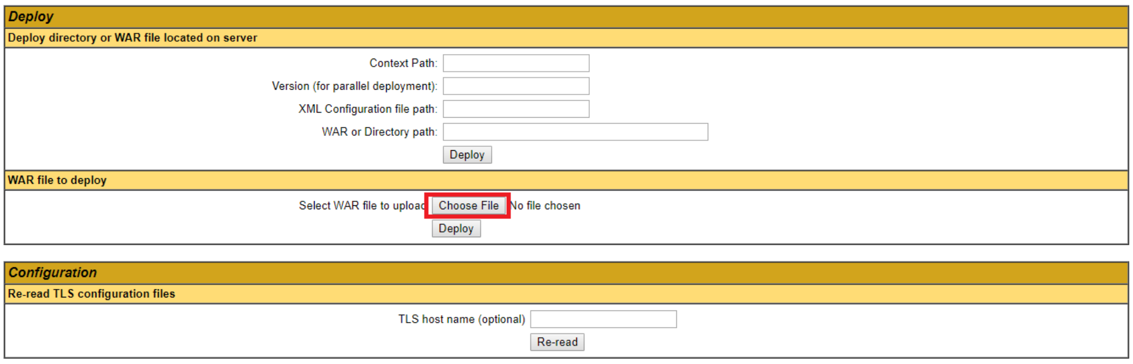
- Open Eclipse.
- Right-click on Project.
- Click on Export.
- Click on the Web in the Export window.
- Select WAR file and click on Next button.
- Enter the destination where the WAR file has to be saved.
- Check the ‘Export source file’ and ‘Overwrite Existing file’ checkboxes.

Step 6.3: Publishing Report on Tomcat

- Open the browser.
- Enter the url: https://localhost:8090 and click on Enter.
- Click on the Manager App button.


localhost:8090
Home Documentation Configuration Examples Wiki Mailing Lists Find Help
Apache Tomcat/9.0.26
If you're seeing this, you've successfully installed Tomcat. Congratulations!
Recommended Reading:
Security Considerations How-To
Manager Application How-To
Clustering/Session Replication How-To
Server Status
Manager App
Host Manager
Developer Quick Start
Tomcat Setup
First Web Application
Realms & AAA
JDBC DataSources
Examples
Servlet Specifications
Tomcat Versions

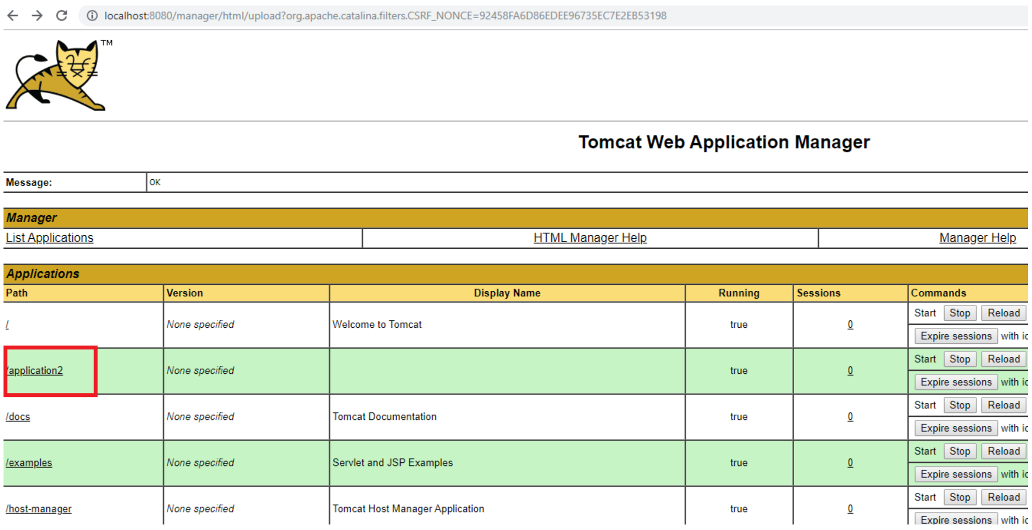
- Move to 'WAR file to deploy' block and click on "Choose" file.



Deploy	
Deploy directory or WAR file located on server	
Context Path:	
Version (for parallel deployment):	
XML Configuration file path:	
WAR or Directory path:	
	Deploy
WAR file to deploy	
Select WAR file to upload	Choose File No file chosen
	Deploy

Configuration	
Re-read TLS configuration files	
TLS host name (optional)	
	Re-read

- Select the WAR file which we have created from Eclipse.
- Click on the 'Deploy' button.
- The filename will be displayed in applications list in Tomcat home page and will look like



localhost:8080/manager/html/upload?org.apache.catalina.filters.CSRF_NONCE=92458FA6D86EDEE96735EC7E2EB53198

Tomcat Web Application Manager

Message:	OK

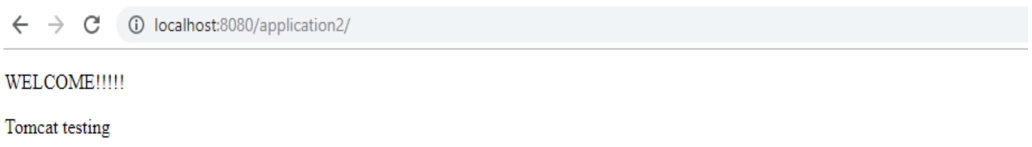
Manager

List Applications	HTML Manager Help	Manager Help

Applications

Path	Version	Display Name	Running	Sessions	Commands
/	None specified	Welcome to Tomcat	true	0	Start Stop Reload Expire sessions with ic
/application2	None specified		true	0	Start Stop Reload Expire sessions with ic
/docs	None specified	Tomcat Documentation	true	0	Start Stop Reload Expire sessions with ic
/examples	None specified	Servlet and JSP Examples	true	0	Start Stop Reload Expire sessions with ic
/host-manager	None specified	Tomcat Host Manager Application	true	0	Start Stop Reload Expire sessions with ic

- Click on the file name which we have deployed.
- The report is displayed in the Tomcat(output), which will look like



localhost:8080/application2/

WELCOME!!!!!

Tomcat testing

7. XML Parsers

Step 7.1: Explaining types of XML parsers

There are mainly three types of XML parsers:

7.1.1 SAX

7.1.2 DOM

7.1.3 Pull parser

Step 7.1.1: SAX

SAX stands for 'Simple API for XML'. It does not create any internal structure. Clients do not know what methods to call. They just override the methods of the API and place his own code inside the method. It is an event-based parser, it works as an event handler in Java.

- Advantages
    - Since it reads each unit of XML, it creates an event so that the calling program can use it.
    - SAX uses what it likes to, by ignoring the bits which it doesn't care about.
    - It is memory efficient.
    - It's very fast and works for huge documents.
- Disadvantages
    - The main disadvantage of SAX is that the Calling program must keep track of everything it might ever need.
    - Since its Event-based, its API is less Intuitive.

Step 7.1.2: DOM

DOM stands for 'Document Object Model'. A DOM Parser creates an internal structure in memory which is a DOM document object and the client applications get information of the original XML document by invoking methods on this document object. DOM Parser has a tree-based structure.

- Advantages
    - It supports both Read and Write operations.
    - It is preferred when there is random access to widely separated parts of the documents required.
    - It builds the entire XML document representation in memory and then hands the calling program the whole chunk of memory.
- Disadvantages

- It consumes more memory since the whole XML document will be loaded into the memory.

Step 7.1.3: Pull Parser

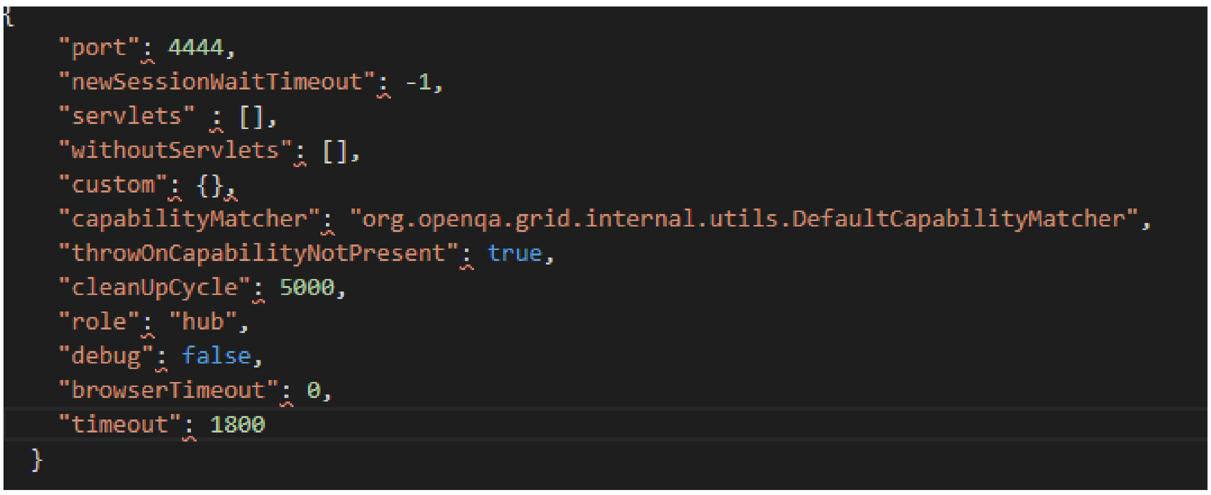
Pull parser waits for the application to come calling. That is, they ask for the next available event, and the application basically loops until it runs out of XML.

- Advantages
    - It is designed to be used with large data sources.
    - Pull parser chooses to skip the events (whole section of the document) which it is not interested in.

8. Grid Configuration Using JSON

Step 1: Configuring the grid hub using JSON

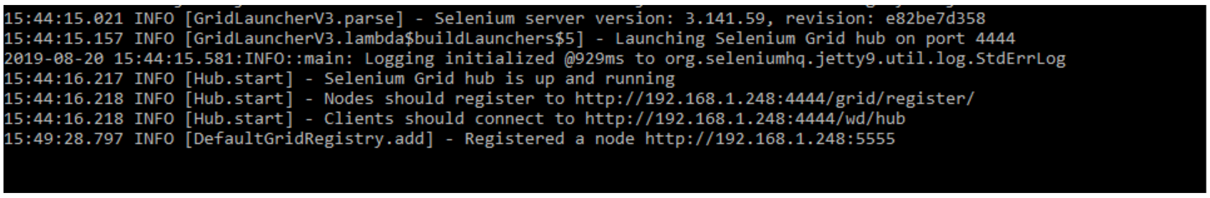
a. Create JSON file for the hub which will look like:



```
{
    "port": 4444,
    "newSessionWaitTimeout": -1,
    "servlets" : [],
    "withoutServlets": [],
    "custom": {},
    "capabilityMatcher": "org.openqa.grid.internal.utils.DefaultCapabilityMatcher",
    "throwOnCapabilityNotPresent": true,
    "cleanUpCycle": 5000,
    "role": "hub",
    "debug": false,
    "browserTimeout": 0,
    "timeout": 1800
  }
```

b. Save it in a folder with a valid name (example: myhub) in which we have saved Selenium standalone Server jar file.
c. Go to the command prompt.
d. Navigate to the folder structure where you have saved the Selenium standalone Server jar file.
e. Type the below command in the command prompt

Java -jar selenium-server-standalone-3.141.59.jar -role hub -hubConfig myhub.json and click on **Enter**. It will look like:

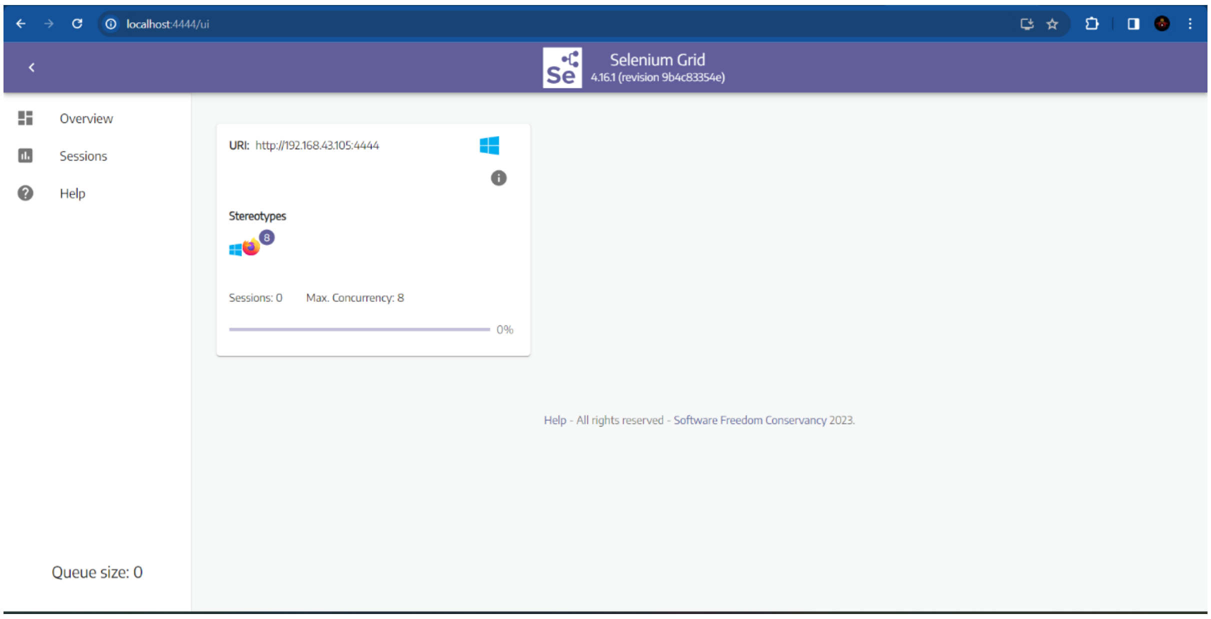


```
15:44:15.021 INFO [GridLauncherV3.parse] - Selenium server version: 3.141.59, revision: e82be7d358
15:44:15.157 INFO [GridLauncherV3.lambda$buildLaunchers$5] - Launching Selenium Grid hub on port 4444
2019-08-20 15:44:15.581:INFO::main: Logging initialized @929ms to org.seleniumhq.jetty9.util.log.StdErrLog
15:44:16.217 INFO [Hub.start] - Selenium Grid hub is up and running
15:44:16.218 INFO [Hub.start] - Nodes should register to http://192.168.1.248:4444/grid/register/
15:44:16.218 INFO [Hub.start] - Clients should connect to http://192.168.1.248:4444/wd/hub
15:49:28.797 INFO [DefaultGridRegistry.add] - Registered a node http://192.168.1.248:5555
```

f. Open the Chrome browser.

g. Enter URL as ‘http://localhost:4444/grid/console’ and click on **Enter.**

h. Grid console page is loaded as below.



URI: http://192.168.43.105:4444

Stereotypes

8

Sessions: 0 Max. Concurrency: 8

0%

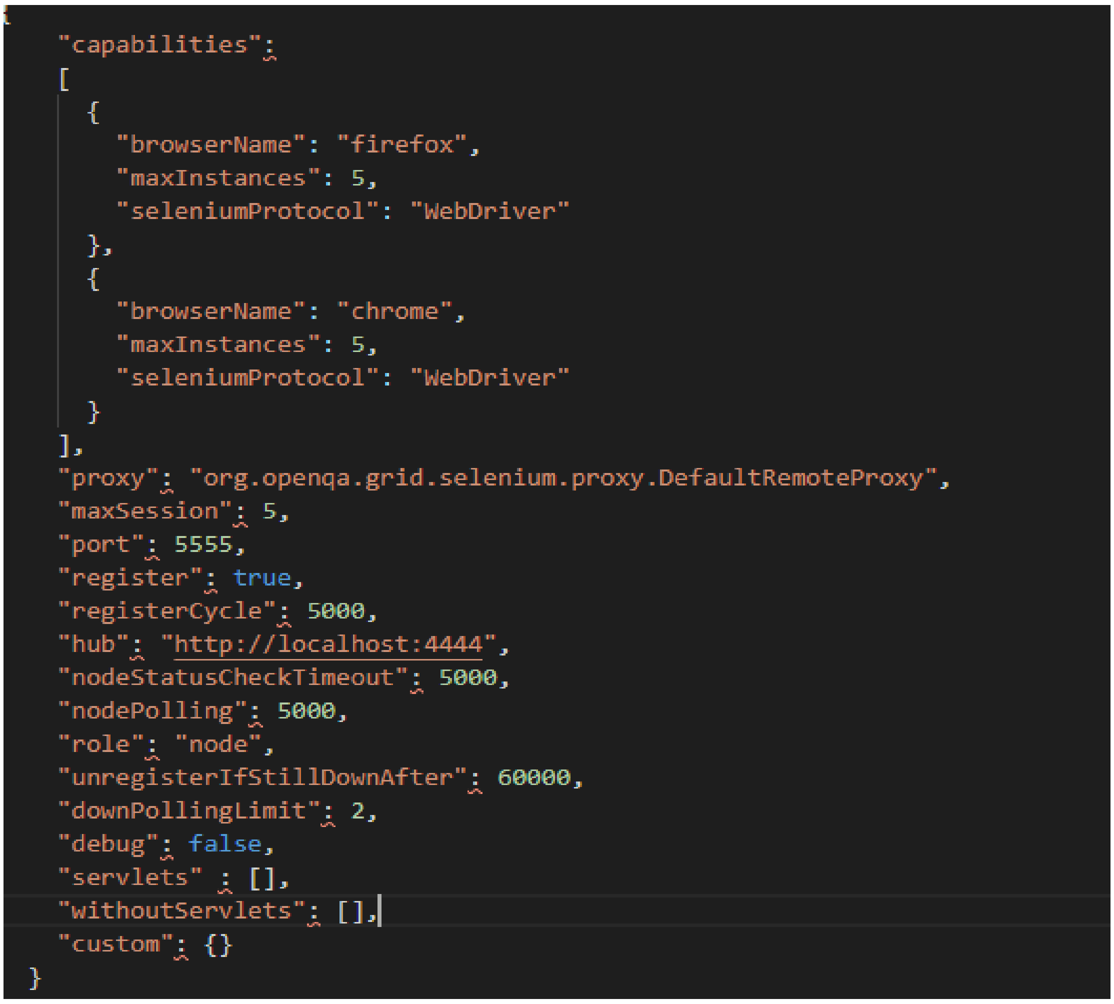
Help - All rights reserved - Software Freedom Conservancy 2023.

Queue size: 0

Step 2: Configuring the grid nodes using JSON

a. Once the Selenium Grid Hub using JSON is configured, the next step is to configure Selenium Grid nodes using JSON.

b. Create a JSON file for node, which will look like:



```
{
  "capabilities":
  [
    {
      "browserName": "firefox",
      "maxInstances": 5,
      "seleniumProtocol": "WebDriver"
    },
    {
      "browserName": "chrome",
      "maxInstances": 5,
      "seleniumProtocol": "WebDriver"
    }
  ],
  "proxy": "org.openqa.grid.selenium.proxy.DefaultRemoteProxy",
  "maxSession": 5,
  "port": 5555,
  "register": true,
  "registerCycle": 5000,
  "hub": "http://localhost:4444",
  "nodeStatusCheckTimeout": 5000,
  "nodePolling": 5000,
  "role": "node",
  "unregisterIfStillDownAfter": 60000,
  "downPollingLimit": 2,
  "debug": false,
  "servlets" : [],
  "withoutServlets": [],
  "custom": {}
}
```

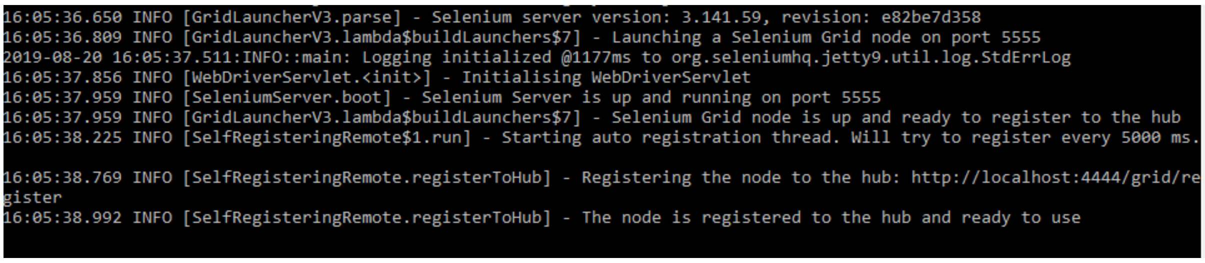
c. Save it in a folder with a valid name (example: mynode) in which we have saved Selenium standalone Server jar file.

d. Open the new command prompt.

e. Navigate to the folder structure where you have saved the Selenium standalone Server jar file.

f. Type the below command in the command prompt

java -Dwebdriver.gecko.driver="geckodriver.exe" -Dwebdriver.chrome.driver="chromedriver.exe" -jar selenium-server-standalone-3.141.59.jar -role node -nodeConfig mynodes.json and click on **Enter** button, which will look like:



```
16:05:36.650 INFO [GridLauncherV3.parse] - Selenium server version: 3.141.59, revision: e82be7d358
16:05:36.809 INFO [GridLauncherV3.lambda$buildLaunchers$7] - Launching a Selenium Grid node on port 5555
2019-08-20 16:05:37.511:INFO::main: Logging initialized @1177ms to org.seleniumhq.jetty9.util.log.StdErrLog
16:05:37.856 INFO [WebDriverServlet.<init>] - Initialising WebDriverServlet
16:05:37.959 INFO [SeleniumServer.boot] - Selenium Server is up and running on port 5555
16:05:37.959 INFO [GridLauncherV3.lambda$buildLaunchers$7] - Selenium Grid node is up and ready to register to the hub
16:05:38.225 INFO [SelfRegisteringRemote$1.run] - Starting auto registration thread. Will try to register every 5000 ms.

16:05:38.769 INFO [SelfRegisteringRemote.registerToHub] - Registering the node to the hub: http://localhost:4444/grid/re
gister
16:05:38.992 INFO [SelfRegisteringRemote.registerToHub] - The node is registered to the hub and ready to use
```

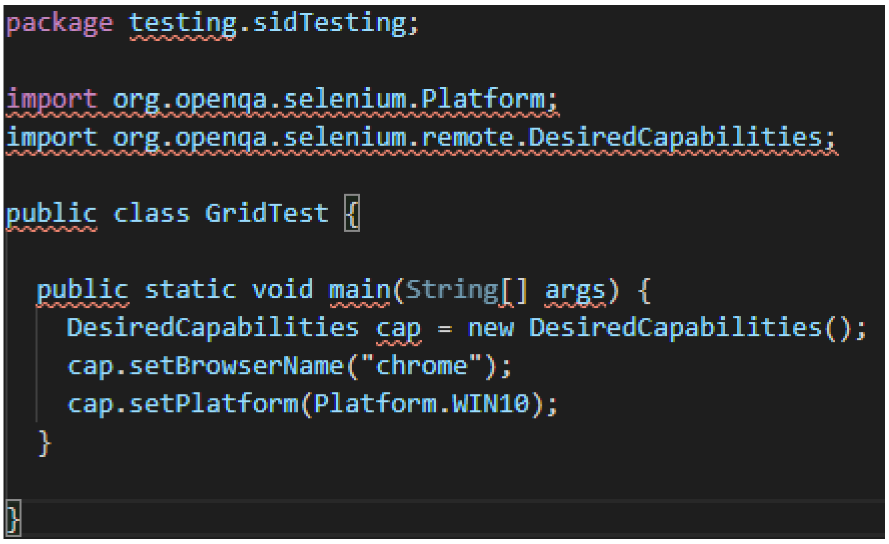
- Open the browser.

- Enter URL as **http://localhost:4444/grid/console** and click on **Enter.**
- The Grid console page will get loaded, which shows **Browsers** by default.
- Click on **Configuration** which shows the configuration details.

9. Demonstrate Running Tests on Selenium Grid on Multiple Browsers .

Step 1: Running the Tests on Grid

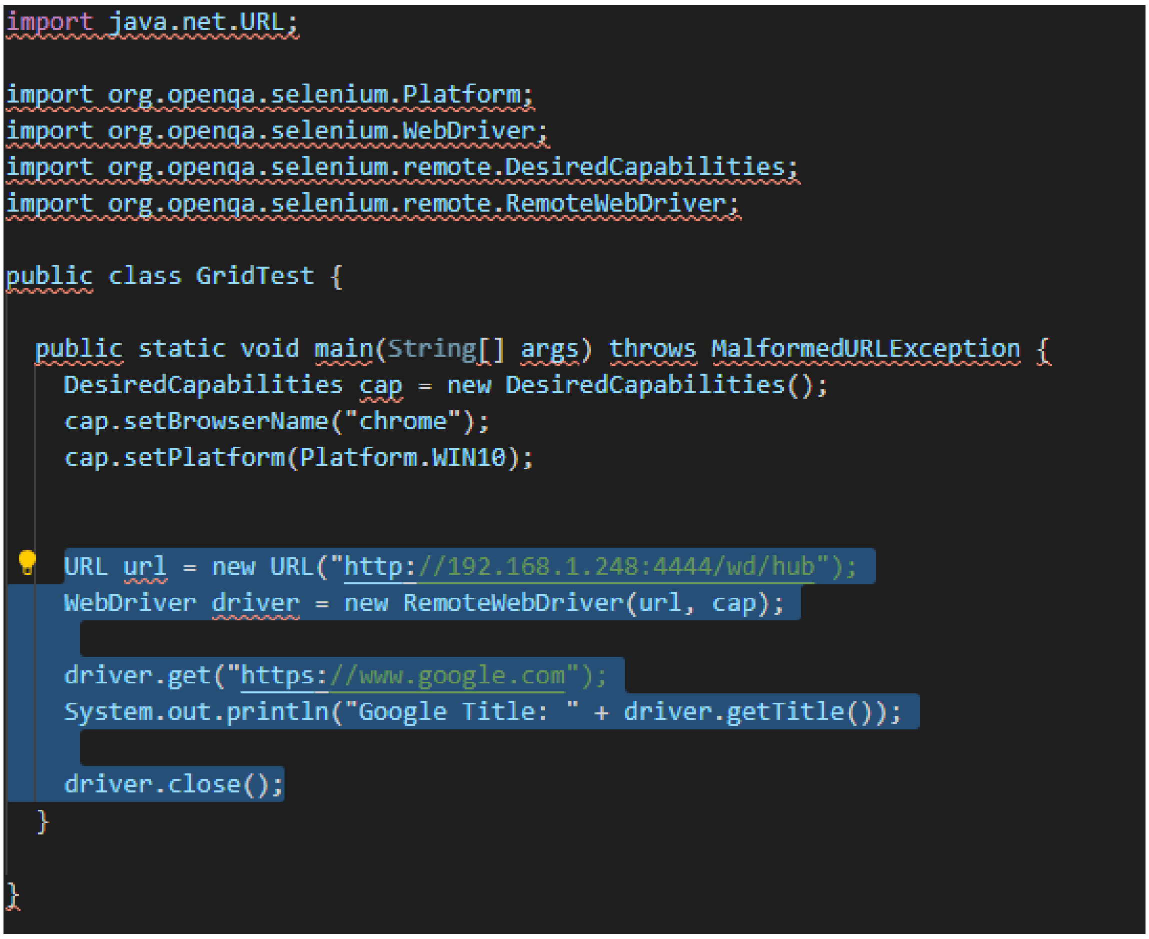
- Open Eclipse.
- Click on **File>New>Other> Class.**
- Give a valid Class name (example: GridTest).
- Check the **public static void main** checkbox and click on **finish**
  , which will then create a blank Java class.
- Write the desired capabilities in the class, which will look like:



```java
package testing.sidTesting;

import org.openqa.selenium.Platform;
import org.openqa.selenium.remote.DesiredCapabilities;

public class GridTest {

  public static void main(String[] args) {
    DesiredCapabilities cap = new DesiredCapabilities();
    cap.setBrowserName("chrome");
    cap.setPlatform(Platform.WIN10);
  }

}
```

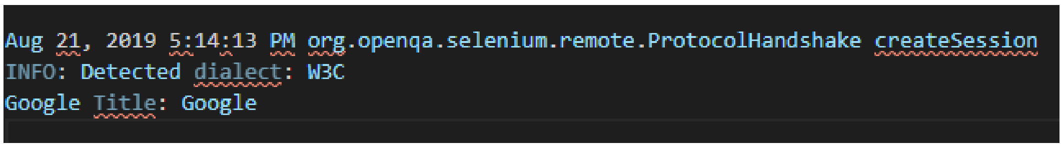
- Start the selenium grid hub in the command prompt using **java -jar selenium-server-standalone-3.141.59.jar -role hub** command.
- Start the Selenium grid node in the command prompt using **java -Dwebdriver.chrome.driver="chromedriver.exe -jar selenium-server-standalone-3.141.59.jar -role node -hub** http://localhost:4444/grid/register command.
- Go to Eclipse and add a statement for remoteWebdriver, which has an implementation of WebDriver, to pass the hub port (`http://192.168.1.248:4444/wd/hub`), and DesiredCapabilities object as parameters.

- Write Selenium code to open the browser and navigate to any web page (example: Google page).



```java
import java.net.URL;

import org.openqa.selenium.Platform;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.remote.DesiredCapabilities;
import org.openqa.selenium.remote.RemoteWebDriver;

public class GridTest {

  public static void main(String[] args) throws MalformedURLException {
    DesiredCapabilities cap = new DesiredCapabilities();
    cap.setBrowserName("chrome");
    cap.setPlatform(Platform.WIN10);


    URL url = new URL("http://192.168.1.248:4444/wd/hub");
    WebDriver driver = new RemoteWebDriver(url, cap);

    driver.get("https://www.google.com");
    System.out.println("Google Title: " + driver.getTitle());

    driver.close();
  }

}
```

- Execute the Java program by right-clicking on the program and navigating to **Run As--> 1 Java Application.**
- This is how it looks like in the Eclipse console.



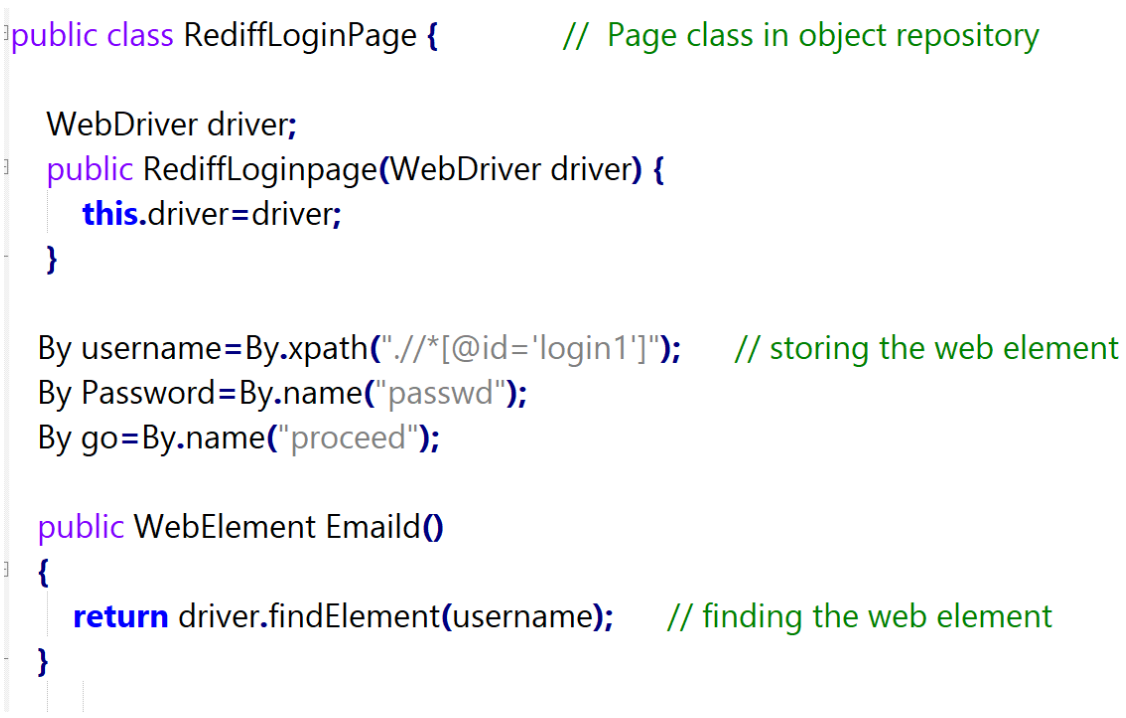
```
Aug 21, 2019 5:14:13 PM org.openqa.selenium.remote.ProtocolHandshake createSession
INFO: Detected dialect: W3C
Google Title: Google
```

- We can see that the capabilities passed through are displayed in both command prompts in the server (hub) as well as in clients (node).

10. Demonstrate page object design pattern in Selenium.

Step 1: Implementing Page Object Model

- This is the basic structure of the Page Object Model (POM), where all Web Elements of the Application Under Test and the method that operate on these Web Elements, are maintained inside a class file.



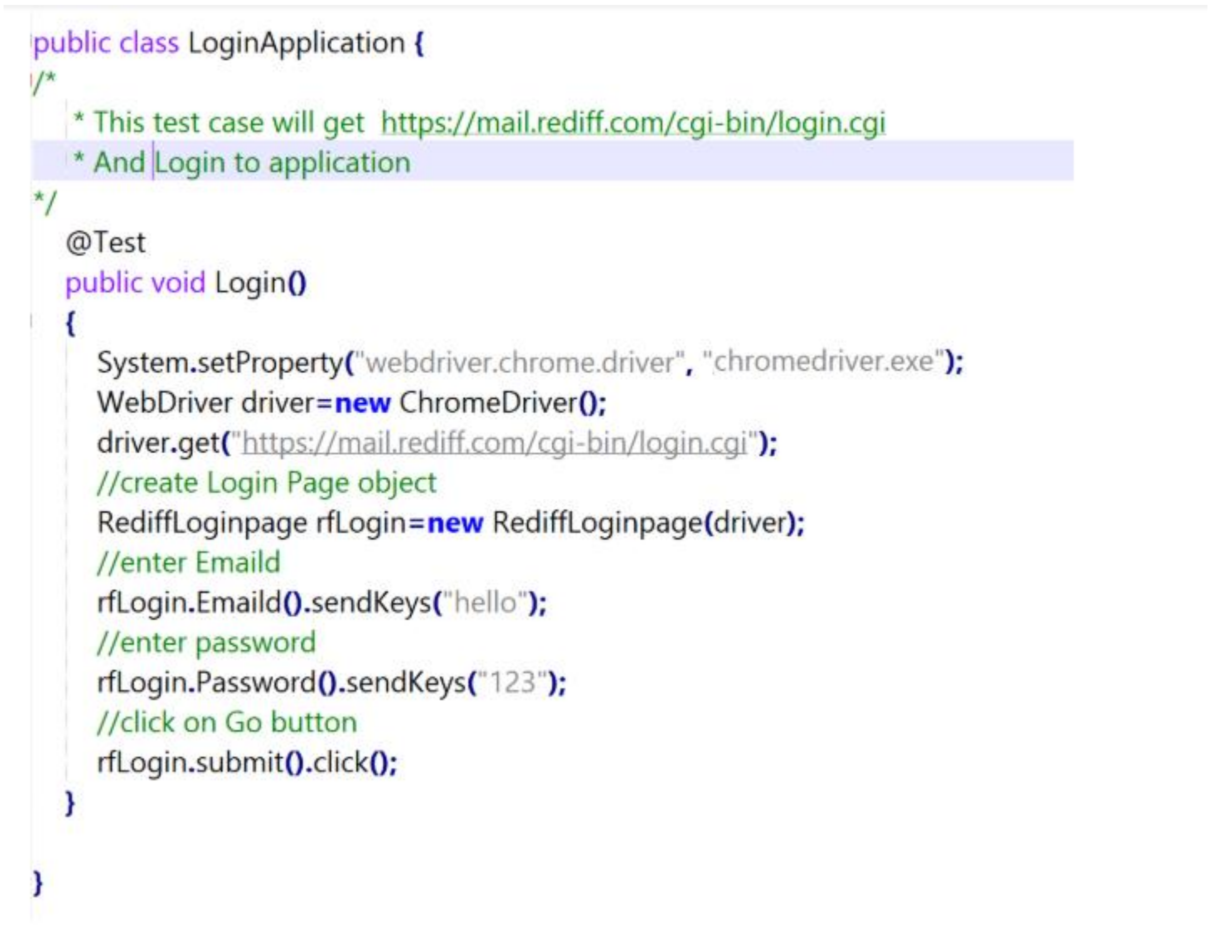
```
public class RediffLoginPage {          //  Page class in object repository

    WebDriver driver;
    public RediffLoginpage(WebDriver driver) {
        this.driver=driver;
    }

   By username=By.xpath(".//*[@id='login1']");      // storing the web element
   By Password=By.name("passwd");
   By go=By.name("proceed");

   public WebElement EmailId()
   {
       return driver.findElement(username);      // finding the web element
   }
```

Step 2: Writing code to implement the Page Object Model using a test case

- Test Case: Log in to Rediff mail. Here we will be dealing with Login page POM and Test script.
- Login Page:

```
public class RediffLoginPage {            //  Page class in object repository

    WebDriver driver;
    public RediffLoginpage(WebDriver driver) {
        this.driver=driver;
    }

   By username=By.xpath(".//*[@id='login1']");     // storing the web element
   By Password=By.name("passwd");
   By go=By.name("proceed");

   public WebElement EmaiId()
   {
       return driver.findElement(username);     // finding the web element
   }

   public WebElement Password()
   {
       return driver.findElement(Password);
   }

   public WebElement submit()
   {
       return driver.findElement(go);
   }

}
```

- Test Case:

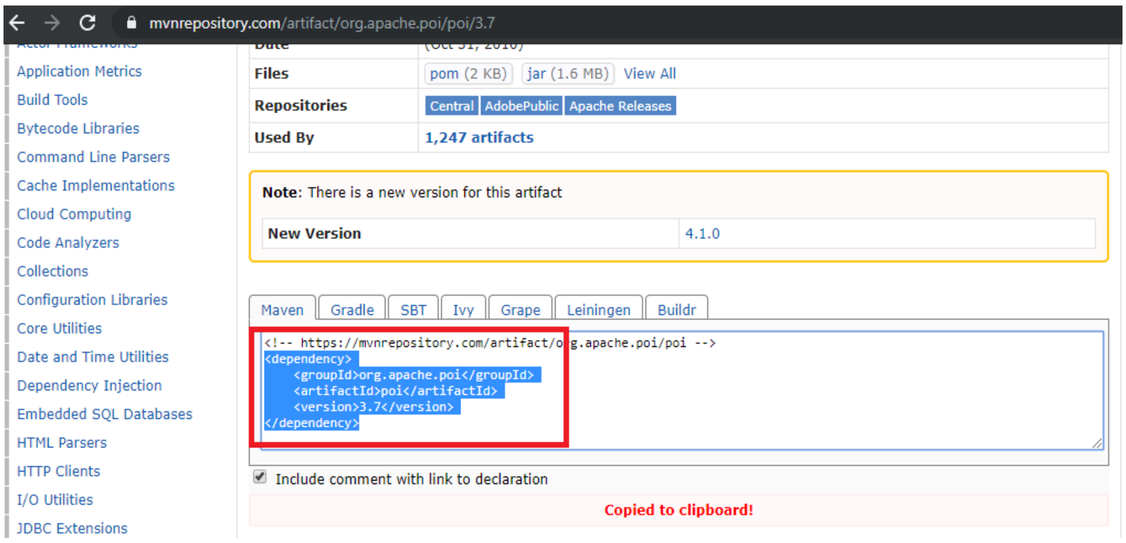

```
public class LoginApplication {
/*
   * This test case will get  https://mail.rediff.com/cgi-bin/login.cgi
   * And Login to application
*/
   @Test
   public void Login()
   {
      System.setProperty("webdriver.chrome.driver", "chromedriver.exe");
      WebDriver driver=new ChromeDriver();
      driver.get("https://mail.rediff.com/cgi-bin/login.cgi");
      //create Login Page object
      RediffLoginpage rfLogin=new RediffLoginpage(driver);
      //enter EmailId
      rfLogin.EmailId().sendKeys("hello");
      //enter password
      rfLogin.Password().sendKeys("123");
      //click on Go button
      rfLogin.submit().click();
   }

}
```

11. Demonstrate how Apache POI is configured in Selenium.

Step 1.12.2: Adding Maven Dependency

a. Go to the Maven Repository: https://mvnrepository.com/artifact/org.apache.poi/poi/3.7

b. Copy the dependency.



mvnrepository.com/artifact/org.apache.poi/poi/3.7

Application Metrics
Build Tools
Bytecode Libraries
Command Line Parsers
Cache Implementations
Cloud Computing
Code Analyzers
Collections
Configuration Libraries
Core Utilities
Date and Time Utilities
Dependency Injection
Embedded SQL Databases
HTML Parsers
HTTP Clients
I/O Utilities
JDBC Extensions

Files	pom (2 KB) jar (1.6 MB) View All
Repositories	Central AdobePublic Apache Releases
Used By	1,247 artifacts

Note: There is a new version for this artifact

New Version	4.1.0

Maven Gradle SBT Ivy Grape Leiningen Buildr

```
<!-- https://mvnrepository.com/artifact/org.apache.poi/poi -->
<dependency>
    <groupId>org.apache.poi</groupId>
    <artifactId>poi</artifactId>
    <version>3.7</version>
</dependency>
```

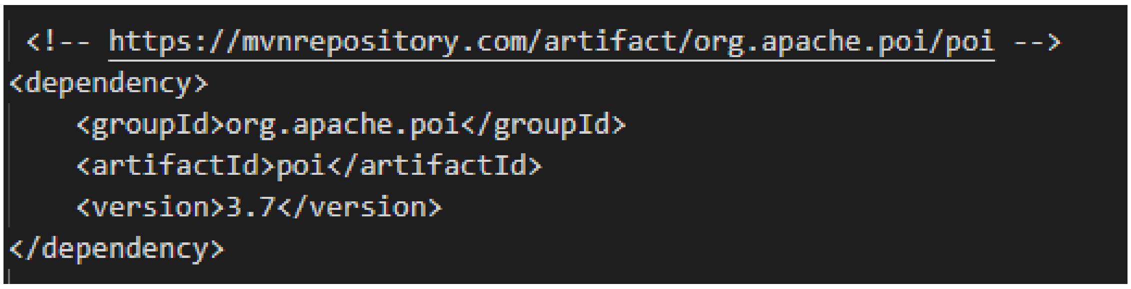
Include comment with link to declaration

Copied to clipboard!

c. Go to Eclipse.

d. Open the Pom.xml file from the project.

e. Copy the dependency in the Pom.xml file:



```
 <!-- https://mvnrepository.com/artifact/org.apache.poi/poi -->
<dependency>
    <groupId>org.apache.poi</groupId>
    <artifactId>poi</artifactId>
    <version>3.7</version>
</dependency>
```

12. Demonstrate how data is read from an Excel sheet in Selenium.

Step 1: Reading the data from the Excel sheet:

- Create an Excel, enter the data, and Save it in a particular location.
- Create a Project.
- Open Eclipse -> File -> New -> Java Project.
- Create a class.
- Right-click on src (default package) -> New -> Class.
- Load the file and specify the path in it:

FileInputStream obj = new FileInputStream ("Path of the excel sheet")

- Load the Workbook:

  Workbook obj1 = WorkbookFactory.create(obj);

- Load the Sheet:

  Sheet obj2 = obj1.getSheet("Sheet1");

- Specify which row you want to read:

Row obj3 = obj2.getRow(1);

- Specify which column to read and the data type:

  Cell obj4 = obj3.getCell(0);

  String st = obj4.getStringCellValue();

- Code in Eclipse will look like this:

```
//Load the file
FileInputStream fis = new FileInputStream("/home/ubuntu/Desktop/Testdata.xlsx");
//Load the Workbook
Workbook wb = WorkbookFactory.create(fis);
//Load Sheet
Sheet sh = wb.getSheet("Sheet1");
//Specify which row you want to read
Row rw = sh.getRow(1);
//Specify which cell want to read and which data type
Cell cel = rw.getCell(0);
 String st = cel.getStringCellValue();
 System.out.println("Username is"+ st);
```

Step 2: Write the data in an Excel sheet

- Create an Excel file, enter the data, and save it in a particular location.
- Create a Project.
- Open Eclipse -> File -> New -> Java Project.
- Create a class.
- Right-click on src (default package) -> New -> Class.
- Load the file and specify the path in it.

  FileInputStream obj = new FileInputStream ("Path of the excel sheet")

- Load the Workbook.

  Workbook obj1 = WorkbookFactory.create(obj);

- Load the Sheet.

  Sheet obj2 = obj1.getSheet(“Sheet1”);

- Specify which row you want to read:

Row obj3 = obj2.getRow(1);

- Specify in which column you want to write:

  Cell obj4 = obj3.getCell(0);

  obj4.setCellValue(“Enter the value”);

- Write the output to a file.

  FileOutputStream obj5 = new FileOutputStream(“Path of the excel sheet”);

  Obj1.write(obj5);

- Code in Eclipse will look like this:

```
  //Load the file
 FileInputStream fis = new
 FileInputStream("/home/ubuntu/Desktop/Testdata.xlsx");
  //Load the Workbook
 Workbook wb = WorkbookFactory.create(fis);
 //Load Sheet
 Sheet sh = wb.getSheet("Sheet1");
 //Specify which row you want to read
 Row rw = sh.getRow(1);
//Specify which row want to read and which data type
 Cell cel = rw.getCell(0);
 cel.setCellValue("John");
//Write the output to a file
 FileOutputStream fout = new
 FileOutputStream("/home/ubuntu/Desktop/Testdata.xlsx");
 wb.write(fout);
```

13. Integrate Selenium with Maven

1. Set Up Maven:

Download and install Java JDK.

Download and install Apache Maven.

Verify Maven installation by running mvn -version in your terminal.

2. Create a Maven Project:

Open your preferred IDE (e.g., Eclipse, IntelliJ IDEA).

Create a new Maven project.

Choose an appropriate group ID, artifact ID, and version for your project.

3. Add Selenium Dependencies:

Open the pom.xml file in your project directory.

Add the Selenium WebDriver dependency within the <dependencies> section. For example:

XML

```
<dependencies>
  <dependency>
    <groupId>org.selenium</groupId>
    <artifactId>selenium-webdriver</artifactId>
    <version>4.4.3</version>
  </dependency>
</dependencies>
```

content_copy

You can add specific browser driver dependencies based on your needs
(e.g., chromedriver, geckodriver).

4. Configure Test Framework (Optional):

While Selenium WebDriver is for browser automation, test frameworks like TestNG or JUnit provide test case organization and reporting.

Add the chosen framework's dependency to the pom.xml file.

5. Write Selenium Test Cases:

Create Java classes for your test cases.

Use the Selenium WebDriver API to interact with web elements.

Leverage the chosen test framework for test organization and execution.

6. Run Tests:

Use Maven commands to run your tests, such as mvn test or mvn verify.

Maven will download necessary dependencies, build your project, and execute the tests.

14. Demonstrate integration of Selenium with Ant

Steps 14.1: Installing Ant

- Ant is already installed in your Practice lab. Refer QA to QE bal guide for Phase 2 for more information.

Steps 14.2: Writing a code for build.xml

build.xml is the most important component of the Ant build tool. For a Java project, all learning, setup, compilation and deployment related tasks are mentioned in this file in an XML format. When we execute this XML file using a command line or any IDE plugin, all instructions written into this file will get executed in a sequential manner.

Let's understand the code within a sample build.xml.

- Project tag is used to mention a project name and basedir attribute. The basedir is the root directory of an application.

```
<project name="YTMonetize" basedir=".">
```

- Property tags are used as variables in the build.xml file to be used in further steps.

```
<property name="build.dir" value="${basedir}/build"/>
                        <property name="external.jars" value=".\resources"/>
        <property name="ytoperation.dir" value="${external.jars}/YTOperation"/>
<property name="src.dir"value="${basedir}/src"/>
```

- Target tags are used as steps that will execute in a sequential order. The Name attribute is the name of the target. You can have multiple targets in a single build.xml.

```
        <target name="setClassPath">
```

- **path** tag is used to bundle all the files logically which are in the common location.

```
<path id="classpath_jars">
```

- **pathelement** tag will set the path to the root of the common location where all the files are stored.

```
<pathelement path="${basedir}/"/>
```

- **pathconvert** tag is used to convert paths of all the common file inside the path tag to the system's classpath format.

```
<pathconvert pathsep=";" property="test.classpath" refid="classpath_jars"/>
```

- **fileset** tag is used to set the classpath for different third-party jars in our project.

```
<fileset dir="${ytoperation.dir}" includes="*.jar"/>
```

- **Echo** tag is used to print the text on the console.

```
<echo message="deleting existing build directory"/>
```

- **Delete** tag will clean the data from the given folder.

```
<delete dir="${build.dir}"/>
```

- **mkdir** tag will create a new directory.

```
<mkdir dir="${build.dir}"/>
```

- **javac** tag is used to compile the java source code and move the .class files to a new folder.

```
    <javac destdir="${build.dir}" srcdir="${src.dir}">
        <classpath refid="classpath_jars"/>
</jalvac>
```

- **jar** tag will create a jar file from the .class files.

```
<jar destfile="${ytoperation.dir}/YTOperation.jar" basedir="${build.dir}">
```

- **manifest** tag will set your main class for execution.

```
<manifest>
        <attribute name="Main-Class" value="test.Main"/>
```

```
</manifest>
```

- 'depends' attribute is used to make a target dependent on another target.

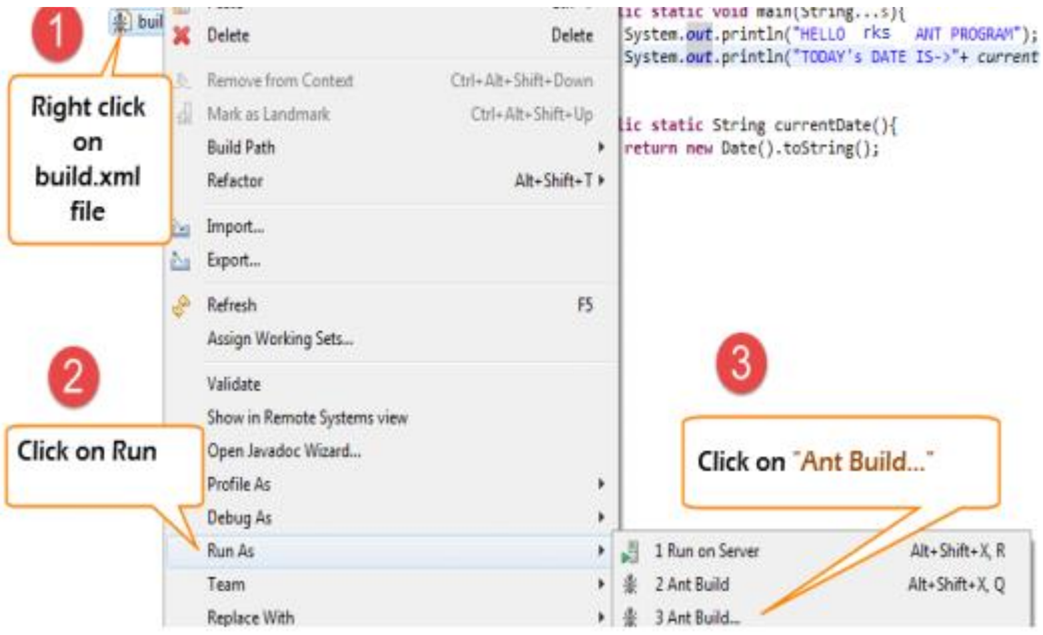
  <target name="run" depends="compile">

- **java** tag will execute the main function from the jar created in the compile target section.

```
<java jar="${ytoperation.dir}/YTOperation.jar" fork="true"/>
```

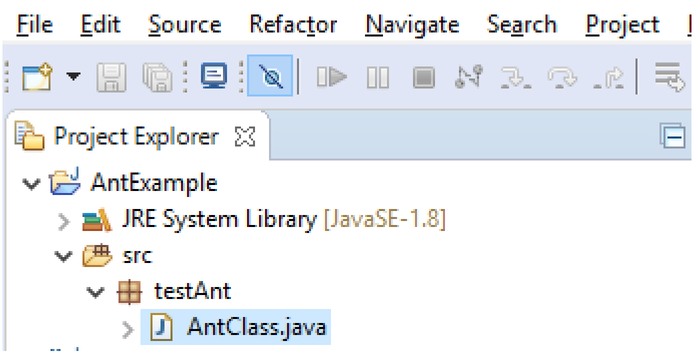
Steps 14.3: Running Ant using Eclipse plugin

- To run Ant from Eclipse, go to build.xml file -> right click on the file -> Run as… -> Build file.



Steps 14.4: Writing a code to implement the functionality of Ant

- We will take a small sample program that will explain the Ant functionality very clearly. Our project structure will look something like –


File Edit Source Refactor Navigate Search Project
Project Explorer
AntExample
JRE System Library [JavaSE-1.8]
src
testAnt
AntClass.java

- Here in this example, we have 4 targets:

    1. Set the classpath for external jars.
    2. Clean the previously compiled code.
    3. Compile the existing java code.
    4. Run the code.

```
AntClass.class

package testAnt;

import java.util.Date;


public class AntClass {

  public static void main(String...s){


                System.out.println("HELLO ANT PROGRAM");


                System.out.println("TODAY's DATE IS->"+ currentDate() );


}

public static String currentDate(){

      return new Date().toString();

      }

}
```

Steps 1.25.5: Writing a build.xml file

```
<?xml version="1.0" encoding="UTF-8"  standalone="no"?>
```

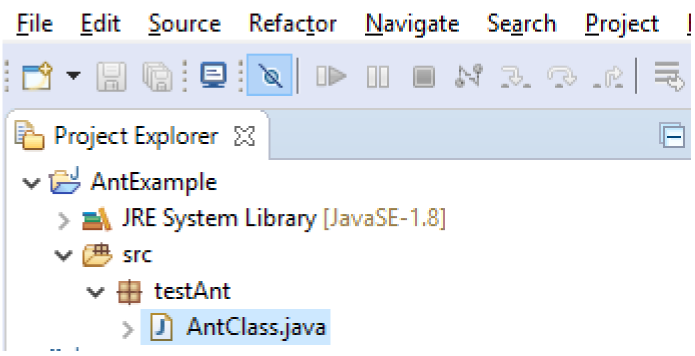
```
<!--Project tag used to mention the project name, and basedir attribute will be the
root directory of the application-->


<project name="YTMonetize" basedir=".">


    <!--Property tags will be used as variables in build.xml file to use in further steps--
>


      <property name="build.dir" value="${basedir}/build"/>


   <property name="external.jars" value=".\resources"/>


      <property name="ytoperation.dir" value="${external.jars}/YTOperation"/>

<property name="src.dir"value="${basedir}/src"/>

<!--Target tags used as steps that will execute in sequential order. name attribute
will be the name  of the target and < a name=OLE_LINK1 >'depends' attribute used
to make one target to depend on another target -->

            <target name="setClassPath">


                                    <path id="classpath_jars">


                                                   <pathelement          path="${basedir}/"/>


                                    </path>

<pathconvert  pathsep=";"property="test.classpath" refid="classpath_jars"/>


</target>

      <target name="clean">


                     <!--echo tag will use to print text on console-->

      <echo message="deleting existing build directory"/>


       <!--delete tag will clean data from given folder-->

                     <delete dir="${build.dir}"/>


                     </target>
```

```
        <target name="compile" depends="clean,setClassPath">

        <echo message="classpath:${test.classpath}"/>

                                    <echo message="compiling........."/>

        <!--mkdir tag will create new director-->

        <mkdir dir="${build.dir}"/>

                  <echo message="classpath:${test.classpath}"/>

                  <echo message="compiling........."/>

 <!--javac tag used to compile java source code and move .class files to a new folder-
->

<javac destdir="${build.dir}" srcdir="${src.dir}">
```

```
                                    <classpath refid="classpath_jars"/>

        </javac>

        <!--jar tag will create jar file from .class files-->

        <jar          destfile="${ytoperation.dir}/YTOperation.jar"basedir="${build.dir}">

              <!--manifest tag will set your main class for execution-->

        <manifest>

        <attribute name="Main-Class" value="testAnt. AntClass"/>

        </manifest>

        </jar>

         </target>

        <target name="run" depends="compile">

<!--java tag will execute main function from the jar created in compile target section-->

<java jar="${ytoperation.dir}/YTOperation.jar"fork="true"/>
```

```
</target>

          </project>
```

Steps 14.6: Executing the TestNG code using Ant


File Edit Source Refactor Navigate Search Project
Project Explorer
AntExample
JRE System Library [JavaSE-1.8]
src
testAnt
AntClass.java

- Here we will create a class with the TestNG methods and set the classpath for Testing in build.xml.
- Now to execute the TestNG method, we will create another testng.xml file and call this file from the build.xml file.

Step 1) We create an **"AntClass.class"** in package **testAnt.**

```
AntClass.class

package testAnt;

import java.util.Date;

import org.testng.annotations.Test;

public class AntClass {

  @Test

        public void AntTestNGMethod(){

                  System.out.println("HELLO ANT PROGRAM");

                  System.out.println("TODAY's DATE IS->"+ currentDate() );

        }
        public static String currentDate(){

                return new Date().toString();

        }
}
```

Step 2) Create a target to load this class in build.xml.

```
<!-- Load testNG and add to the class path of application -->
        <target name="loadTestNG" depends="setClassPath">
<!—using taskdef  tag we can add a task to run on the current project. In
below  line, we are adding testing task in this project. Using testing task here
now we    can run testing code using the ant script -->
        <taskdef resource="testngtasks" classpath="${test.classpath}"/>
</target>
```

Step 3) Create testng.xml

```
testng.xml

<?xml version="1.0"encoding="UTF-8"?>
<!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd">
<suite name="YT"thread-count="1">
                              <test name="TestNGAnt">
                              <classes>
                                <class name="testAnt. AntClass">
        </class>
</classes>
</test>
</suite>
```

Step 4) Create a Target in build.xml to run this TestNG code.

```
<target name="runTestNGAnt" depends="compile">
```

```
<!-- testng tag will be used to execute testng code using corresponding
testng.xml    file. Here classpath attribute is setting classpath for testng's jar to
the project-->

            <testng classpath="${test.classpath};${build.dir}">

<!—xmlfileset tag is used here to run testng's code using testing.xml file. Using

includes tag we are mentioning path to testing.xml file-->

             <xmlfileset dir="${basedir}" includes="testng.xml"/>

</testng>
```

Step 5) The complete build.xml will look like:

```
<?xml version="1.0"encoding="UTF-8"standalone="no"?>

<!--Project tag used to mention the project name, and basedir attribute will be
the root directory of the application-->

                        <project name="YTMonetize" basedir=".">

<!--Property tags will be used as variables in build.xml file to use in further
steps-->

                        <property name="build.dir"value="${basedir}/build"/>

<!-- put  testng related jar in the resource  folder -->

                <property name="external.jars" value=".\resource"/>

                        <property name="src.dir" value="${basedir}/src"/>

<!--Target tags used as steps that will execute in sequential order. Name
attribute will be the name of the target and 'depends' attribute used to make
one target to depend on another target-->

<!-- Load testNG and add to the class path of application -->

      <target name="loadTestNG"depends="setClassPath">

            <taskdef resource="testngtasks"classpath="${test.classpath}"/>

                        </target>

                        <target name="setClassPath">

                             <path id="classpath_jars">

                        <pathelement path="${basedir}/"/>

                                    <fileset dir="${external.jars}" includes="*.jar"/>

      </path>
```

```
     <pathconvert pathsep=";"property="test.classpath"refid="classpath_jars"/>

            </target>

            <target name="clean">

         <!--echo tag will use to print text on console-->

                      <echo message="deleting existing build directory"/>

          <!--delete tag will clean data from given folder-->

                      <delete dir="${build.dir}"/>

                        </target>

<target name="compile"depends="clean,setClassPath,loadTestNG">

                  <echo message="classpath:${test.classpath}"/>

                      <echo message="compiling........."/>

                             <!--mkdir tag will create new director-->

                              <mkdir dir="${build.dir}"/>

                        <echo message="classpath:${test.classpath}"/>

                                    <echo message="compiling........."/>

<!--javac tag used to compile java source code and move .class files to a new
folder-->

                 <javac destdir="${build.dir}"srcdir="${src.dir}">

                    <classpath refid="classpath_jars"/>

                        </javac>

 </target>

<target name="runTestNGAnt"depends="compile">

<!-- testng tag will be used to execute testng code using corresponding
testng.xml file -->

            <testng classpath="${test.classpath};${build.dir}">

          <xmlfileset dir="${basedir}"includes="testng.xml"/>

            </testng>

</target></project>
```

Steps 14.7: Integrating Ant with Selenium WebDriver

- We have learned that by using Ant we can put all the third party jars in a particular location in the system and set their path for our project.

- Using this method, we are setting all the dependencies of our project in a single place and making it more reliable for compilation, execution, and deployment.
- Similarly, for our testing projects, using Selenium, we can easily mention Selenium dependency in build.xml and we don't need to add a classpath for it manually in our application.
- So, now you can ignore the below-mentioned traditional way to set classpaths for our project.

Steps 14.8: Modifying previous examples

We are going to modify the previous example now.

Step 1) Set the property Selenium.jars to the Selenium related jar in the resource folder.

```
<property name="selenium.jars" value=".\selenium"/>
```

Step 2) In the target **setClassPath**, add the Selenium files.

```
<target name="setClassPath">

                <path id="classpath_jars">

                        <pathelement path="${basedir}/"/>

                        <fileset dir="${external.jars}" includes="*.jar"/>

                   <!-- selenium jar added here -->

                                <fileset dir="${selenium.jars}" includes="*.jar"/>

       </path>
```

Step 3) Complete build.xml:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>


<!--Project tag used to mention the project name, and basedir attribute will
be the root directory of the application-->

                                    <project name="YTMonetize" basedir=".">

              <!--Property tags will be used as variables in build.xml file to use in
further steps-->

         <property name="build.dir" value="${basedir}/build"/>

     <!-- put  testng related jar in the resource  folder -->

              <property name="external.jars" value=".\resource"/>

<!-- put  selenium related jar in resource  folder -->
```

```
    <property name="selenium.jars" value=".\selenium"/>

          <property name="src.dir" value="${basedir}/src"/>

<!--Target tags used as steps that will execute in sequential order. name
attribute will be the name of the target and 'depends' attribute used to
make one target to depend on another target-->

     <!-- Load testNG and add to the class path of application -->

      <target name="loadTestNG" depends="setClassPath">

          <taskdef resource="testngtasks" classpath="${test.classpath}"/>

                          </target>

<target name="setClassPath">

                 <path id="classpath_jars">

                                                         <pathelement path="${basedir}/"/>

          <fileset dir="${external.jars}" includes="*.jar"/>

                                          <!-- selenium jar added here -->

                    <fileset dir="${selenium.jars}"includes="*.jar"/>

       </path>

   <pathconvert pathsep=";" property="test.classpath"
refid="classpath_jars"/>

</target>

<target name="clean">

<!--echo tag will use to print text on console-->

             <echo message="deleting existing build directory"/>

                        <!--delete tag will clean data from given folder-->

                                       <delete dir="${build.dir}"/>

                                                         </target>

<target name="compile" depends="clean,setClassPath,loadTestNG">

        <echo message="classpath:${test.classpath}"/>

              <echo message="compiling.........."/>

       <!--mkdir tag will create new director-->

                 <mkdir dir="${build.dir}"/>

                          <echo message="classpath:${test.classpath}"/>

                          <echo message="compiling.........."/>
```

```
<!--javac tag used to compile java source code and move .class files to new
folder-->

    <javac destdir="${build.dir}"srcdir="${src.dir}">

           <classpath refid="classpath_jars"/>

          </javac>

          </target>

          <target name="runTestNGAnt" depends="compile">

<!-- testng tag will be used to execute testng code using corresponding
testng.xml file -->

                                        <testng
classpath="${test.classpath};${build.dir}">

             <xmlfileset dir="${basedir}" includes="testng.xml"/>

                         </testng>

          </target>

</project>
```

Step 4) Now change the previously created class **AntClass.java** with the new code.

```
AntClass.java:

package testAnt;

import java.util.List;

import org.openqa.selenium.By;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.WebElement;

import org.openqa.selenium.firefox.FirefoxDriver;

import org.testng.annotations.Test;


public class AntClass {

    @Test

                   public void TestNGMethod(){

         WebDriver driver = new FirefoxDriver();
```

```
                driver.get("http://demo.guru99.com/test/home/");

 List<WebElement> listAllCourseLinks =
driver.findElements(By.xpath("//div

[@class='canvas-middle']//a"));


      for(WebElement webElement : listAllCourseLinks) {

              System.out.println(webElement.getAttribute("href"));

                }

              }

}
```

Step 5) Now, you can run your test.

15. Listeners - Context, iTest, iTestResults

Step 15.1: Using iTestListener:

- Listener is an interface that modifies the TestNG behavior.
- Listener listens to the event defined in the Selenium and behaves accordingly.
- It is used in Selenium by implementing Listeners Interface.
- iTestListener is used in Selenium to generate logs or customize the TestNG reports.
- iTestListener has the following methods:
    a. **onStart:** onStart method is called when any Test starts.
    b. **onTestSuccess:** onTestSuccess method is called on the success of any Test.
    c. **onTestFailure:** onTestFailure method is called on the failure of any test.
    d. **onTestSkipped:** onTestSkipped method is called when any test gets skipped.
    e. **onTestFailedButWithinSuccessPercentage:** onTestFailedButWithinSuccessPercentage method is called each time Test fails but within the success percentage.
    f. **onFinish:** onFinish method is called after all the tests are executed.
- Open Eclipse and create a Project.
- Create a Listener class that will implement iTestListener.
- The code in Eclipse will look like:

```
package test.testing;

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

public class ListenersTest implements ITestListener {
```

```
public void onFinish(ITestContext Result) {
System.out.println(Result.getName()+"case finished");


}
public void onStart(ITestContext Result) {
// TODO Auto-generated method stub
}

public void onTestFailedButWithinSuccessPercentage(ITestResult Result) {
// TODO Auto-generated method stub


}

public void onTestFailure(ITestResult Result) {
// TODO Auto-generated method stub
System.out.println("The name of the testcase failed is :"+Result.getName());
}

public void onTestSkipped(ITestResult Result) {
// TODO Auto-generated method stub
System.out.println("The name of the testcase Skipped is :"+Result.getName());
}

public void onTestStart(ITestResult Result) {
// TODO Auto-generated method stub
System.out.println(Result.getName()+" test case started");
}

public void onTestSuccess(ITestResult Result) {
// TODO Auto-generated method stub
System.out.println("The name of the testcase passed is :"+Result.getName());
}

}
```

- Create a Java class and implement a Listener class in this.
- The code in Eclipse will look like:

```
package test.testing;

import org.openqa.selenium.By;
import org.testng.Assert;
import org.testng.annotations.Listeners;
import org.testng.annotations.Test;

@Listeners(test.testing.ListenersTest.class)

    public class Home extends Baseclass {

@Test
public void clickOnCategory()
{
driver.findElement(By.xpath("//div[@id='navigation']/a[1]")).click();
System.out.println("Clicked on links");
}
```

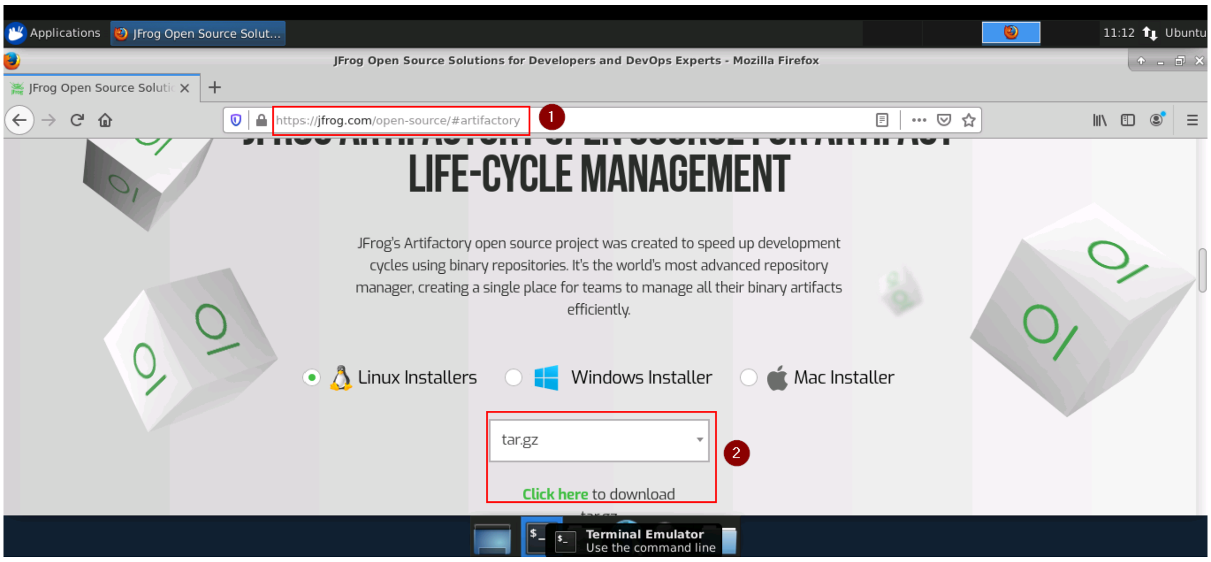
Step 15.2: Running the code

- Run the code through Eclipse.

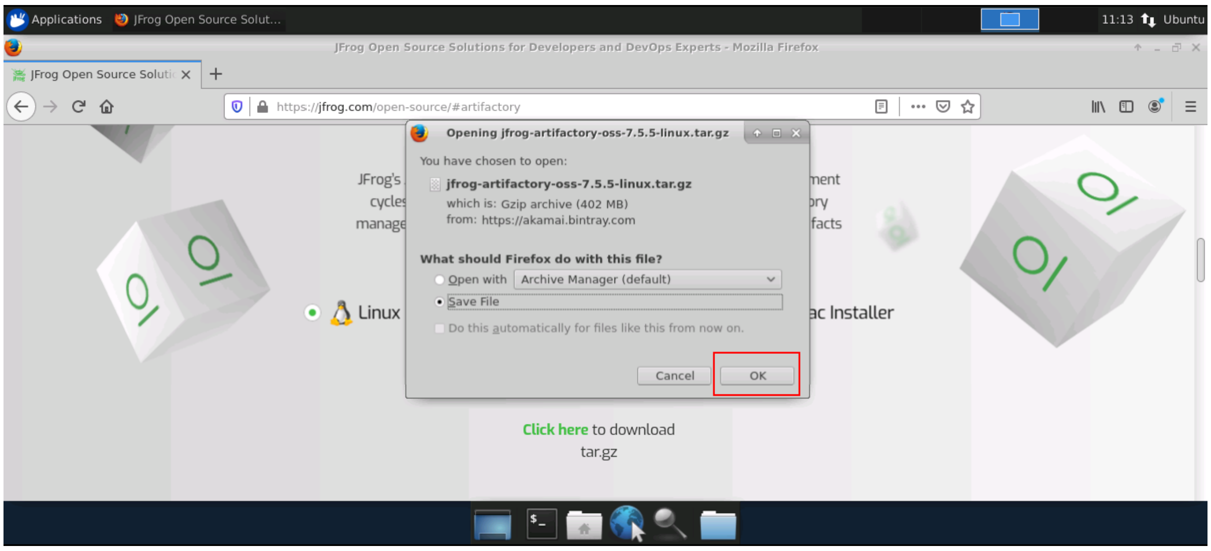
16. Install Artifactory

Step 16.1: Installing Artifactory

- Go to this link: https://jfrog.com/open-source/#artifactory.
- Download the tar.gz file as shown in the following screenshot



JFrog's Artifactory open source project was created to speed up development cycles using binary repositories. It's the world's most advanced repository manager, creating a single place for teams to manage all their binary artifacts efficiently.


Applications
JFrog Open Source Solutions for Developers and DevOps Experts - Mozilla Firefox
https://jfrog.com/open-source/#artifactory
Cancel
OK
Click here to download
tar.gz

Opening jfrog-artifactory-oss-7.5.5-linux.tar.gz

You have chosen to open:

jfrog-artifactory-oss-7.5.5-linux.tar.gz

which is: Gzip archive (402 MB)

from: https://akamai.bintray.com

What should Firefox do with this file?

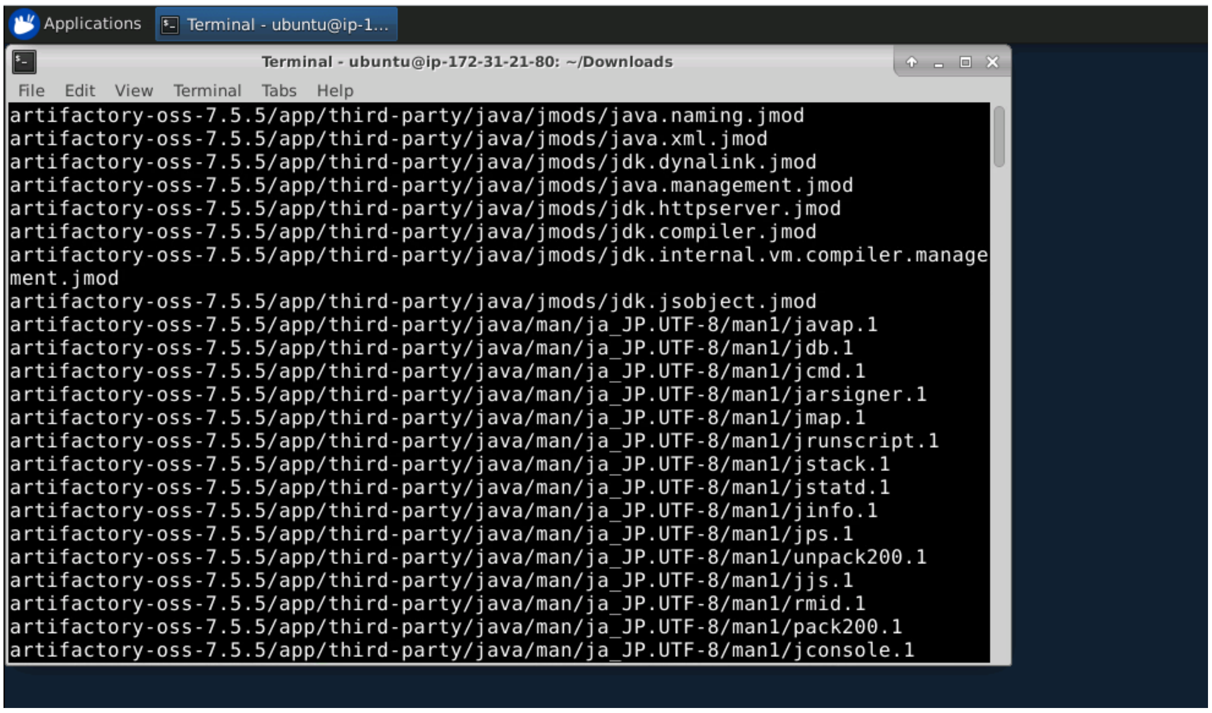
Open with Archive Manager (default) [✓ Save File]

Do this automatically for files like this from now on.

- Use the following command to extract the tar file.

  cd Downloads

  tar -xvf jfrog-artifactory-oss-7.5.5-linux.tar.gz

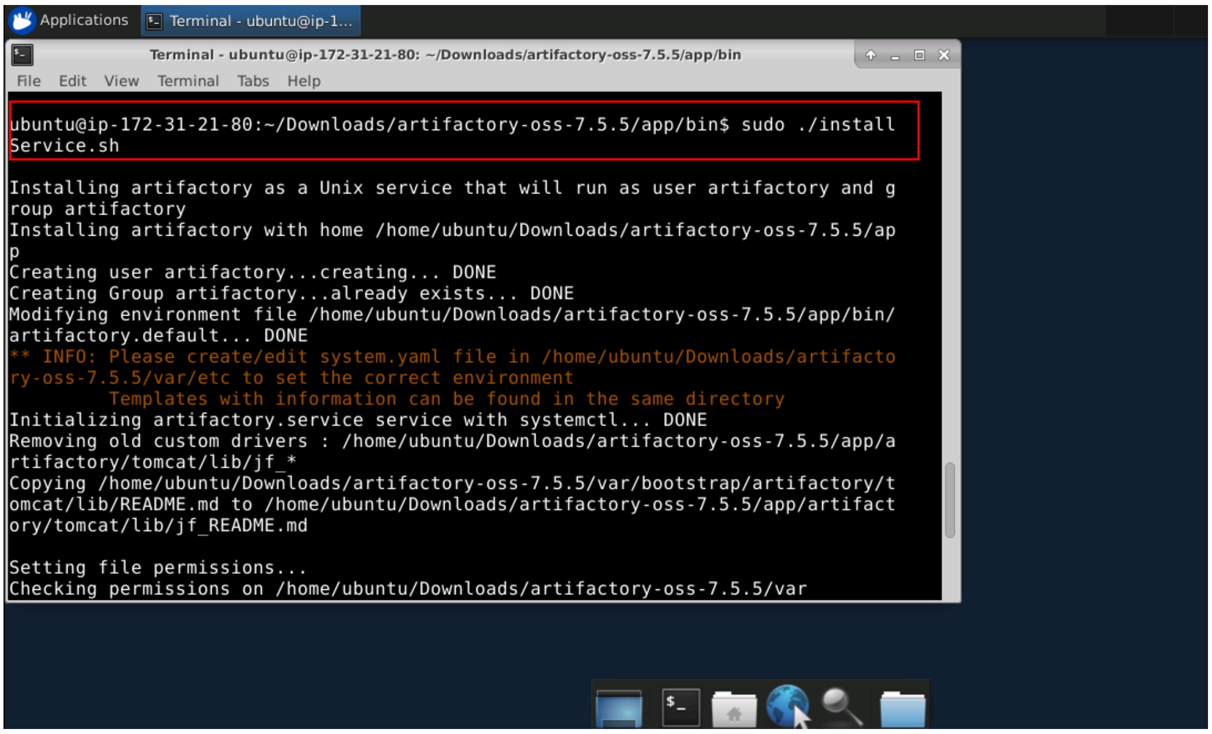

Applications
Terminal - ubuntu@ip-1...
Terminal - ubuntu@ip-172-31-21-80: ~/Downloads
File Edit View Terminal Tabs Help

```
artifactory-oss-7.5.5/app/third-party/java/jmods/java.naming.jmod
artifactory-oss-7.5.5/app/third-party/java/jmods/java.xml.jmod
artifactory-oss-7.5.5/app/third-party/java/jmods/jdk.dynalink.jmod
artifactory-oss-7.5.5/app/third-party/java/jmods/java.management.jmod
artifactory-oss-7.5.5/app/third-party/java/jmods/jdk.httpserver.jmod
artifactory-oss-7.5.5/app/third-party/java/jmods/jdk.compiler.jmod
artifactory-oss-7.5.5/app/third-party/java/jmods/jdk.internal.vm.compiler.manage
ment.jmod
artifactory-oss-7.5.5/app/third-party/java/jmods/jdk.jsobject.jmod
artifactory-oss-7.5.5/app/third-party/java/man/ja_JP.UTF-8/man1/javap.1
artifactory-oss-7.5.5/app/third-party/java/man/ja_JP.UTF-8/man1/jdb.1
artifactory-oss-7.5.5/app/third-party/java/man/ja_JP.UTF-8/man1/jcmd.1
artifactory-oss-7.5.5/app/third-party/java/man/ja_JP.UTF-8/man1/jarsigner.1
artifactory-oss-7.5.5/app/third-party/java/man/ja_JP.UTF-8/man1/jmap.1
artifactory-oss-7.5.5/app/third-party/java/man/ja_JP.UTF-8/man1/jrunscript.1
artifactory-oss-7.5.5/app/third-party/java/man/ja_JP.UTF-8/man1/jstack.1
artifactory-oss-7.5.5/app/third-party/java/man/ja_JP.UTF-8/man1/jstatd.1
artifactory-oss-7.5.5/app/third-party/java/man/ja_JP.UTF-8/man1/jinfo.1
artifactory-oss-7.5.5/app/third-party/java/man/ja_JP.UTF-8/man1/jps.1
artifactory-oss-7.5.5/app/third-party/java/man/ja_JP.UTF-8/man1/unpack200.1
artifactory-oss-7.5.5/app/third-party/java/man/ja_JP.UTF-8/man1/jjs.1
artifactory-oss-7.5.5/app/third-party/java/man/ja_JP.UTF-8/man1/rmid.1
artifactory-oss-7.5.5/app/third-party/java/man/ja_JP.UTF-8/man1/pack200.1
artifactory-oss-7.5.5/app/third-party/java/man/ja_JP.UTF-8/man1/jconsole.1
```

- The tar file will be extracted in a directory named artifactory-oss-7.5.5
- Open the command prompt and go to the bin of the Artifactory folder and run the following commands:

  cd Downloads/artifactory-oss-7.5.5/artifactory/app/bin

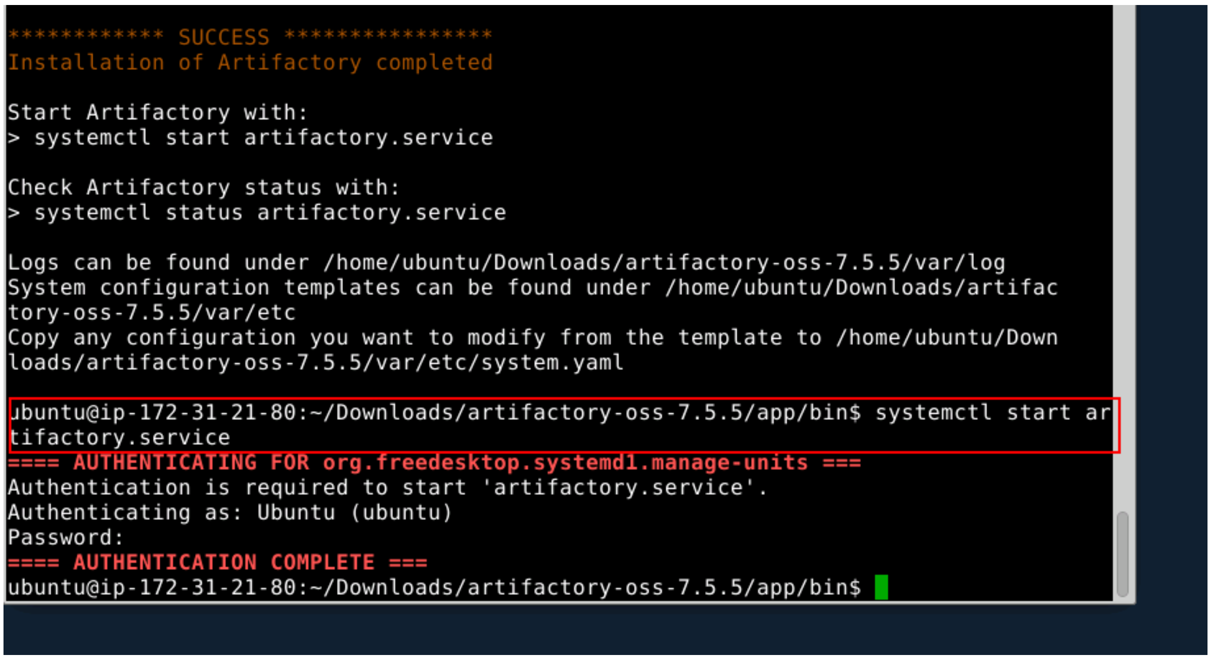
  sudo ./installService.sh


Applications
Terminal - ubuntu@ip-1...
Terminal - ubuntu@ip-172-31-21-80: ~/Downloads/artifactory-oss-7.5.5/app/bin
File Edit View Terminal Tabs Help

```
ubuntu@ip-172-31-21-80:~/Downloads/artifactory-oss-7.5.5/app/bin$ sudo ./install
Service.sh

Installing artifactory as a Unix service that will run as user artifactory and g
roup artifactory
Installing artifactory with home /home/ubuntu/Downloads/artifactory-oss-7.5.5/ap
p
Creating user artifactory...creating... DONE
Creating Group artifactory...already exists... DONE
Modifying environment file /home/ubuntu/Downloads/artifactory-oss-7.5.5/app/bin/
artifactory.default... DONE
** INFO: Please create/edit system.yaml file in /home/ubuntu/Downloads/artifacto
ry-oss-7.5.5/var/etc to set the correct environment
         Templates with information can be found in the same directory
Initializing artifactory.service service with systemctl... DONE
Removing old custom drivers : /home/ubuntu/Downloads/artifactory-oss-7.5.5/app/a
rtifactory/tomcat/lib/jf_*
Copying /home/ubuntu/Downloads/artifactory-oss-7.5.5/var/bootstrap/artifactory/t
omcat/lib/README.md to /home/ubuntu/Downloads/artifactory-oss-7.5.5/app/artifact
ory/tomcat/lib/jf_README.md

Setting file permissions...
Checking permissions on /home/ubuntu/Downloads/artifactory-oss-7.5.5/var
```

- Now, run the command:

  systemctl start artifactory.service

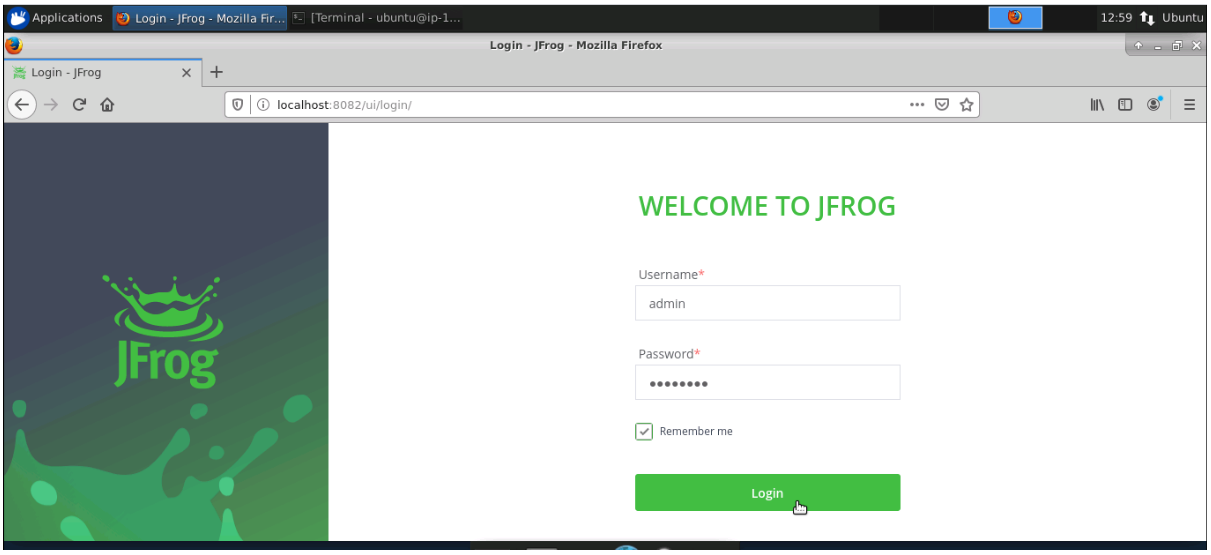


```
************ SUCCESS ****************
Installation of Artifactory completed

Start Artifactory with:
> systemctl start artifactory.service

Check Artifactory status with:
> systemctl status artifactory.service

Logs can be found under /home/ubuntu/Downloads/artifactory-oss-7.5.5/var/log
System configuration templates can be found under /home/ubuntu/Downloads/artifac
tory-oss-7.5.5/var/etc
Copy any configuration you want to modify from the template to /home/ubuntu/Down
loads/artifactory-oss-7.5.5/var/etc/system.yaml

ubuntu@ip-172-31-21-80:~/Downloads/artifactory-oss-7.5.5/app/bin$ systemctl start ar
tifactory.service
==== AUTHENTICATING FOR org.freedesktop.systemd1.manage-units ===
Authentication is required to start 'artifactory.service'.
Authenticating as: Ubuntu (ubuntu)
Password:
==== AUTHENTICATION COMPLETE ===
ubuntu@ip-172-31-21-80:~/Downloads/artifactory-oss-7.5.5/app/bin$
```

- Now, go to the browser type- localhost:8081.
- Enter the following details:

  ID : admin

  Password: password


Applications
Login - JFrog - Mozilla Fir...
[Terminal - ubuntu@ip-1...
12:59
Ubuntu
Login - JFrog - Mozilla Firefox
Login - JFrog
localhost:8082/ui/login/
JFrog

WELCOME TO JFROG

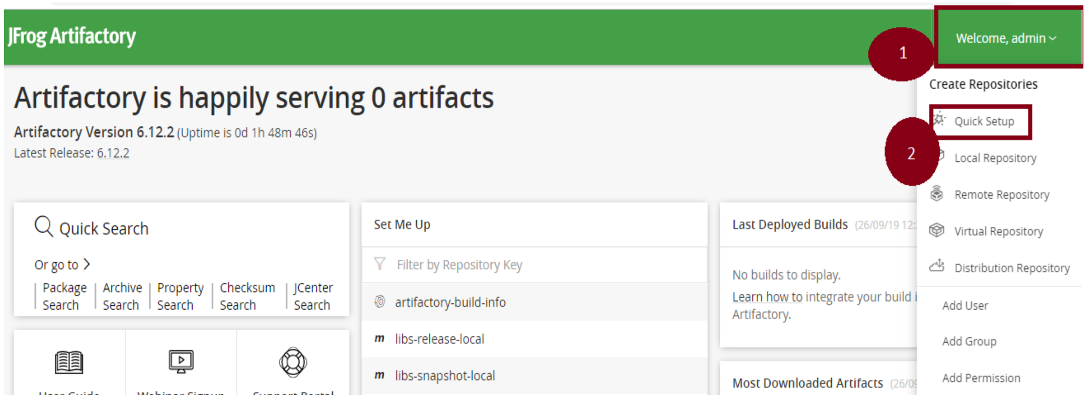
Username*: admin

Password*: ••••••••

✓ Remember me

Login

- Click on 'Welcome Admin.'
- Click on 'Quick Setup.'
- Create a maven repository.



Artifactory is happily serving 0 artifacts

Artifactory Version 6.12.2 (Uptime is 0d 1h 48m 46s)

Latest Release: 6.12.2

artifactory-build-info

libs-release-local

libs-snapshot-local

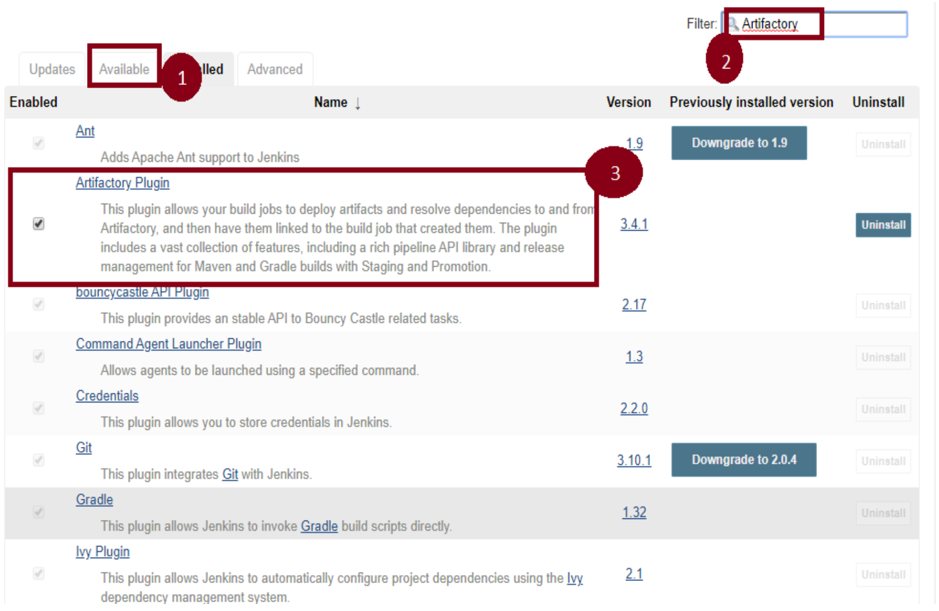
No builds to display.

Learn how to integrate your build i Artifactory.

Step 16.2: Setting up Artifactory with Jenkins

- Jenkins is already installed in your practice lab. Refer to QA to QE lab guide for Phase 2 for more information.
- Open command line and run the following command:

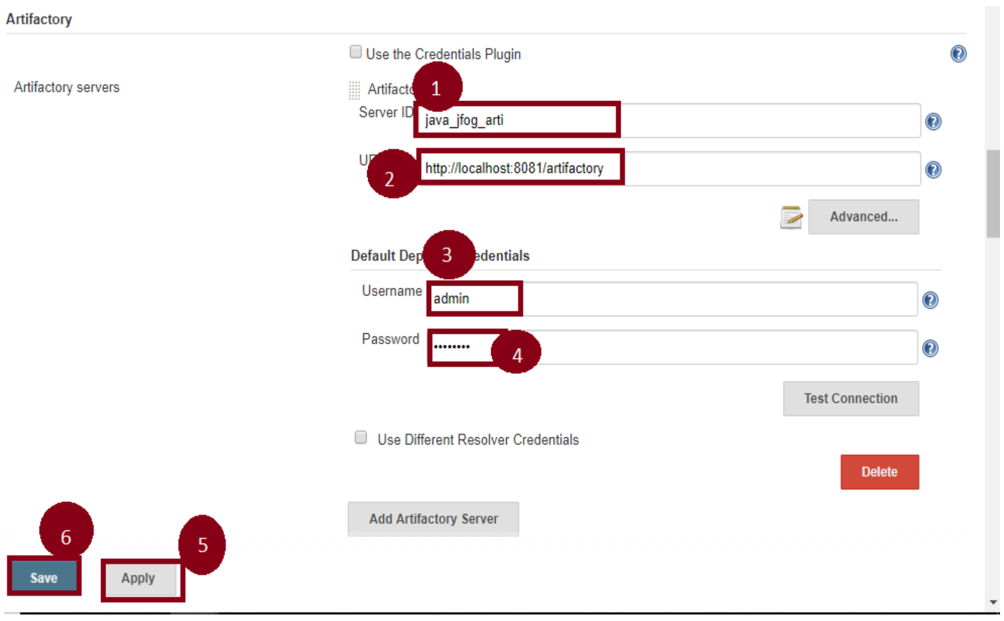
  sudo less /var/lib/jenkins/secrets/initialAdminPassword
- Copy the password displayed in the command line
- Go to localhost:8080 in the browser and paste the password in the given field
- Click on install suggested plugins
- Go to 'Manage Jenkins.'
- Click on 'Manage Plugin.'
- Select the available tab.
- Search for Artifactory.



Enabled	Name ↓	Version	Previously installed version	Uninstall
✓	Ant Adds Apache Ant support to Jenkins	1.9	Downgrade to 1.9	Uninstall
✓	Artifactory Plugin This plugin allows your build jobs to deploy artifacts and resolve dependencies to and fron Artifactory, and then have them linked to the build job that created them. The plugin includes a vast collection of features, including a rich pipeline API library and release management for Maven and Gradle builds with Staging and Promotion.	3.4.1		Uninstall
✓	bouncycastle API Plugin This plugin provides an stable API to Bouncy Castle related tasks.	2.17		Uninstall
✓	Command Agent Launcher Plugin Allows agents to be launched using a specified command.	1.3		Uninstall
✓	Credentials This plugin allows you to store credentials in Jenkins.	2.2.0		Uninstall
✓	Git This plugin integrates Git with Jenkins.	3.10.1	Downgrade to 2.0.4	Uninstall
✓	Gradle This plugin allows Jenkins to invoke Gradle build scripts directly.	1.32		Uninstall
✓	Ivy Plugin This plugin allows Jenkins to automatically configure project dependencies using the Ivy dependency management system.	2.1		Uninstall

- Again, go to 'Manage Jenkins.'

- Go to ‘Configure System.’
- There is an option Artifactory.
- Select ‘Add Artifactory.’
- Pass the artifactory localhost URL.
- Now, pass the credentials.
- Click on ‘Apply’ and ‘Save.’


Artifactory

Use the Credentials Plugin

Artifactory servers

Server ID: java_jfog_arti

URL: http://localhost:8081/artifactory

Default Deployer Credentials

Username: admin

Password: ••••••••

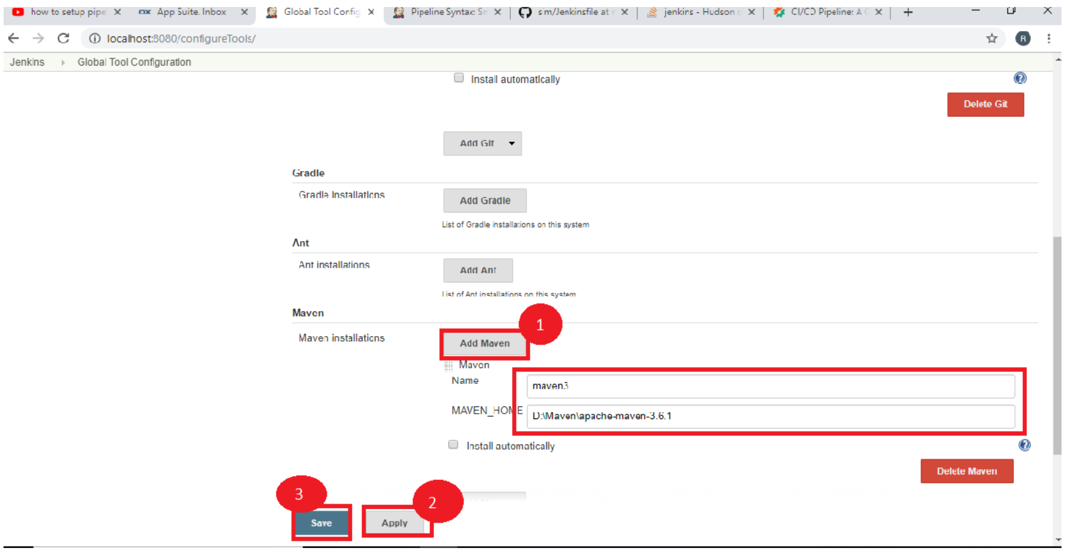
Use Different Resolver Credentials

17. Build and Configure CI/CD Pipeline with Maven Project

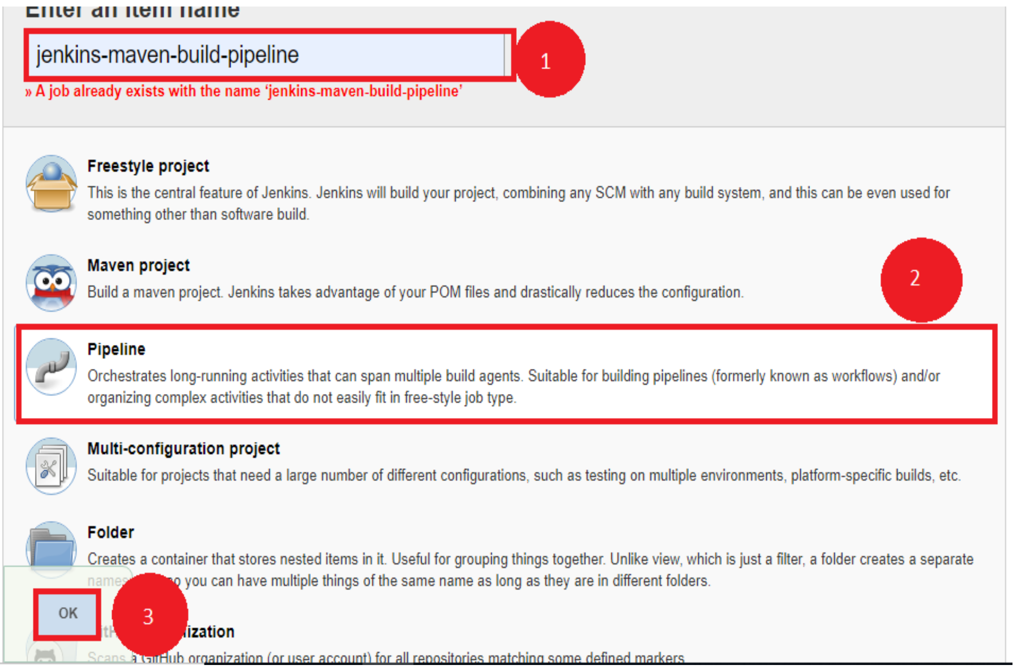
Step 17.1: Setting up a Pipeline

- Jenkins is already set up in your practice lab. Refer to the lab guide for phase 2 for more information.
- Open Jenkins- locahhost:8080.
- Click on **Manage Jenkins.**
- Select **Global Tool Configuration.**

- Set the path of **Java and Maven.**
- Then, **Apply and Save.**



- Select a **New Item.**
- Name the job: **Jenkins-maven-build-pipeline.**
- Select the **Pipeline.**
- Now, click on **OK.**



Enter an item name
jenkins-maven-build-pipeline
1
» A job already exists with the name 'jenkins-maven-build-pipeline'
Freestyle project
This is the central feature of Jenkins. Jenkins will build your project, combining any SCM with any build system, and this can be even used for something other than software build.
Maven project
Build a maven project. Jenkins takes advantage of your POM files and drastically reduces the configuration.
2
Pipeline
Orchestrates long-running activities that can span multiple build agents. Suitable for building pipelines (formerly known as workflows) and/or organizing complex activities that do not easily fit in free-style job type.
Multi-configuration project
Suitable for projects that need a large number of different configurations, such as testing on multiple environments, platform-specific builds, etc.
Folder
Creates a container that stores nested items in it. Useful for grouping things together. Unlike view, which is just a filter, a folder creates a separate name... so you can have multiple things of the same name as long as they are in different folders.
OK
3
...ization
Scans a GitHub organization (or user account) for all repositories matching some defined markers.

Step 17.2: Writing a Jenkins file

- Create a job that needs to be configured.
- Click on **Pipeline.**

- Select the definition as- **Pipeline script from SCM.**
- Pass the GitHub URL.
- Now, we need to write the Jenkins file inside that git file.

```
node{
 stage('SCM Checkout'){
git 'https://github.com/Autamation/sim.git'
}
            stage('Compile-Package'){
   def mvnHome = tool name: 'maven3', type: 'maven'
      bat "${mvnHome}/bin/mvn package"
 bat 'mvn package'
            }
            }
```

- Now, click on 'Apply' and 'Save.'
- Click on **Build Now.**

18. Build and Configure CI/CD Pipeline with Selenium WebDriver

Step 18.1: Foking the git repository

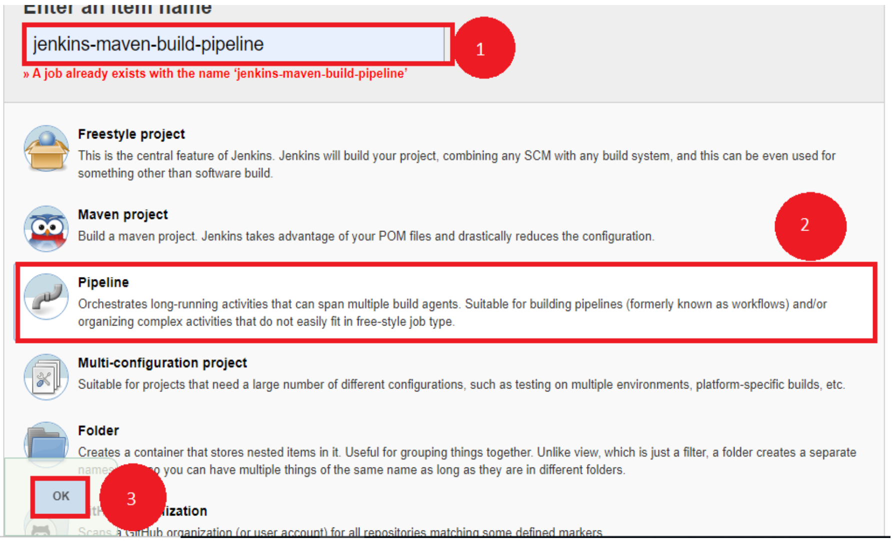
- Fork the following repository

  https://github.com/canindit75/JenkinsDemo

Step 18.2: Create a Jenkins pipeline job

- Java 1.8 is already installed in your practice lab. Refer to QA to QE lab guide for Phase 1 for more information.
- Jenkins.war file is already present in your practice lab in cd /usr/share/jenkins directory.
- Go to jenkins.war location Start the Jenkins by using command on command prompt:**java -jar jenkins.war.**
- Open browser and type **localhost:8080.**
- Enter the password.
- Create a job.

- Pass a name.
- Select **Pipeline.**
- Click on Ok.



Enter an item name

jenkins-maven-build-pipeline

» A job already exists with the name 'jenkins-maven-build-pipeline'

Freestyle project
This is the central feature of Jenkins. Jenkins will build your project, combining any SCM with any build system, and this can be even used for something other than software build.

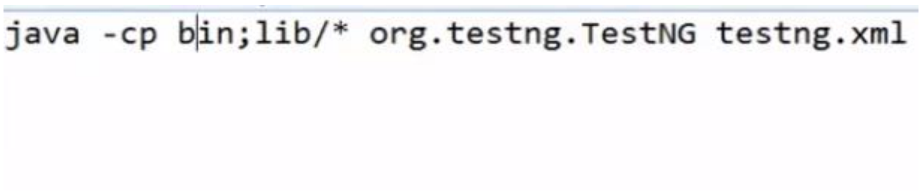
Maven project
Build a maven project. Jenkins takes advantage of your POM files and drastically reduces the configuration.

Pipeline
Orchestrates long-running activities that can span multiple build agents. Suitable for building pipelines (formerly known as workflows) and/or organizing complex activities that do not easily fit in free-style job type.

Multi-configuration project
Suitable for projects that need a large number of different configurations, such as testing on multiple environments, platform-specific builds, etc.

Folder
Creates a container that stores nested items in it. Useful for grouping things together. Unlike view, which is just a filter, a folder creates a separate names[illegible]o you can have multiple things of the same name as long as they are in different folders.

- Create a text file name it **run.sh** in your lab and keep the below given code in it.



```
java -cp bin;lib/* org.testng.TestNG testng.xml
```

- Give executable permission to **run.sh** using the commands below:

chmod 755 run.sh

chmod 777 run.sh

- Push **run.sh in your repository** under master branch.

git push <reponame> master

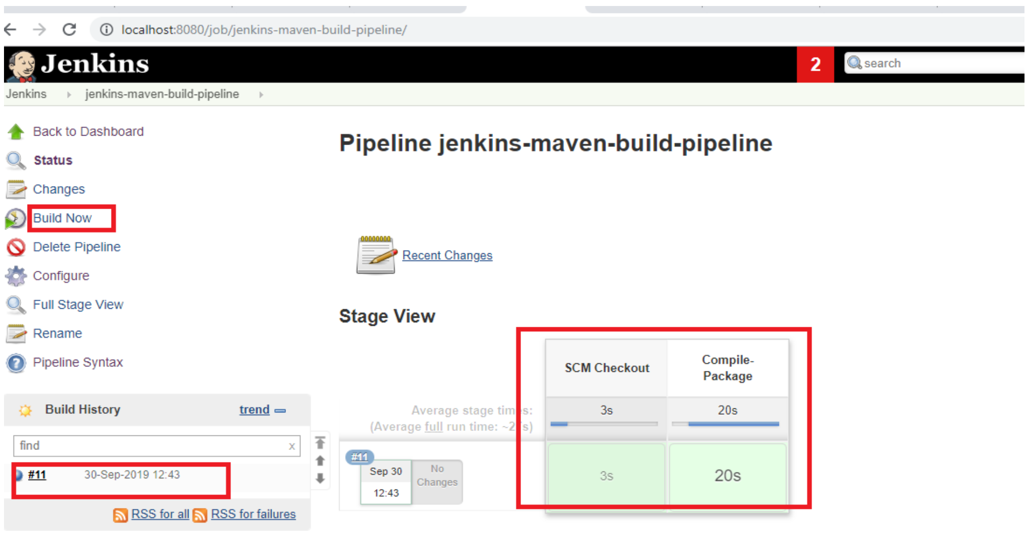
git status

- Go to Jenkins pipeline job.
- Write a groovy script in the pipeline.

```
node {
  def mvnHome
  stage('Preparation') { // for display purposes
    // Get some code from a GitHub repository
    git 'https://github.com/jglick/simple-maven-project-with-tests.git'
    // Get the Maven tool.
    // ** NOTE: This 'M3' Maven tool must be configured
    // **       in the global configuration.
    mvnHome = tool 'maven3'
  }
  stage('Build') {
    // Run the maven build
    withEnv(["MVN_HOME=$mvnHome"]) {
      if (isUnix()) {
        sh '"$MVN_HOME/bin/mvn" -Dmaven.test.failure.ignore clean package'
      } else {
        sh ‘"%MVN_HOME%\bin\mvn" -Dmaven.test.failure.ignore clean package’
      }
    }
  }
  stage('Results') {
    junit '**/target/surefire-reports/TEST-*.xml'
    archiveArtifacts 'target/*.jar'
  }}
```

- Click on Apply and Save.

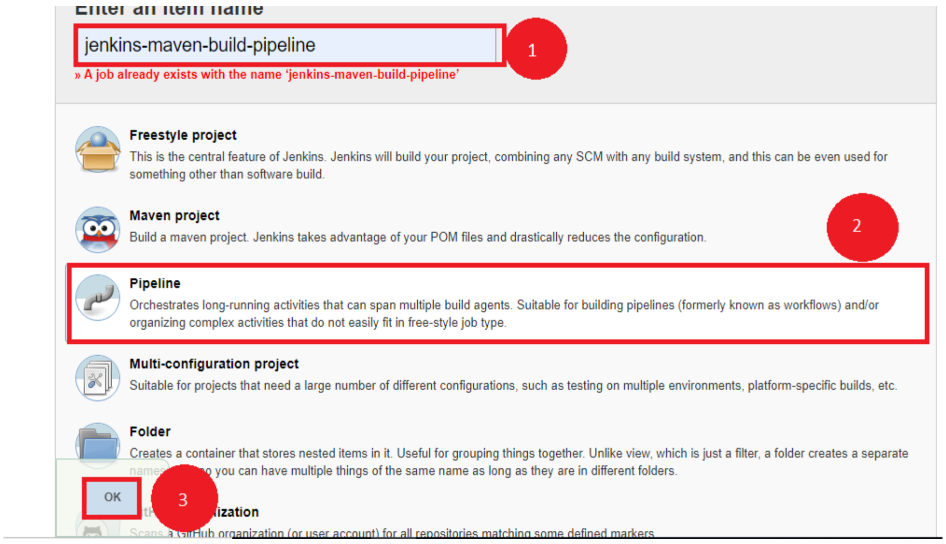
- Click on Build now.



19. Selenium with Jenkins

Step 19.1: Creating a Jenkins Pipeline job

- Java 1.8 is already installed in your practice lab.
- Jenkins.war file is already present in your practice lab in directory /usr/share/jenkins.
- Go to the jenkins.war location. Now, start Jenkins by using the following command on command prompt: **java -jar jenkins.war.**
- Open your browser and type- **localhost:8080.**
- Enter the password.
- Create a job.
- Enter the name.
- Select the **Pipeline.**
- Now, click on OK.


Enter an item name

jenkins-maven-build-pipeline

» A job already exists with the name 'jenkins-maven-build-pipeline'

Freestyle project
This is the central feature of Jenkins. Jenkins will build your project, combining any SCM with any build system, and this can be even used for something other than software build.

Maven project
Build a maven project. Jenkins takes advantage of your POM files and drastically reduces the configuration.

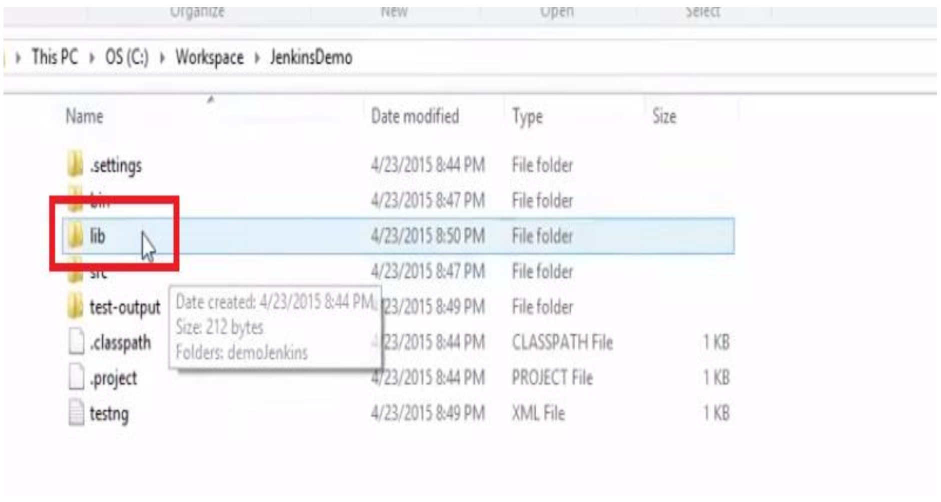
Pipeline
Orchestrates long-running activities that can span multiple build agents. Suitable for building pipelines (formerly known as workflows) and/or organizing complex activities that do not easily fit in free-style job type.

Multi-configuration project
Suitable for projects that need a large number of different configurations, such as testing on multiple environments, platform-specific builds, etc.

Folder
Creates a container that stores nested items in it. Useful for grouping things together. Unlike view, which is just a filter, a folder creates a separate names... so you can have multiple things of the same name as long as they are in different folders.

Step 19.2: Integrating Selenium WebDriver with Jenkins

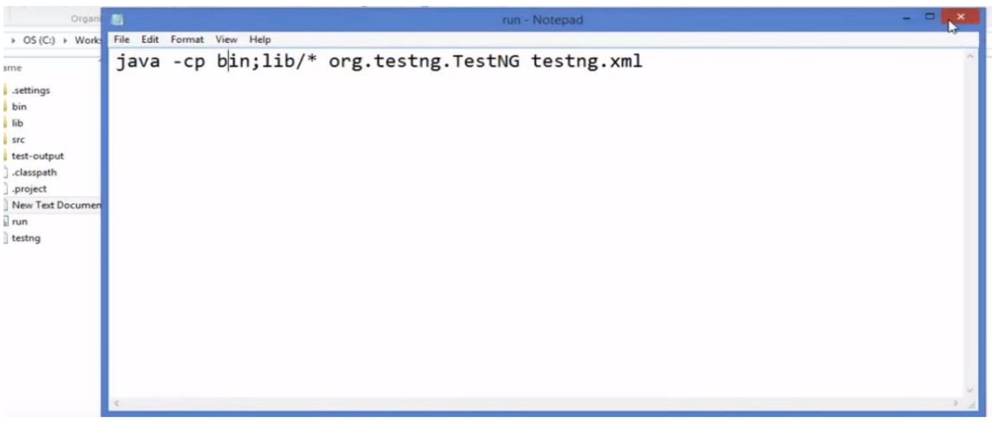
- Go to the Selenium script location.
- Add all the Selenium libraries.


This PC › OS (C:) › Workspace › JenkinsDemo

Name	Date modified	Type	Size
.settings	4/23/2015 8:44 PM	File folder	
bin	4/23/2015 8:47 PM	File folder	
lib	4/23/2015 8:50 PM	File folder	
src	4/23/2015 8:47 PM	File folder	
test-output	4/23/2015 8:49 PM	File folder	
.classpath	4/23/2015 8:44 PM	CLASSPATH File	1 KB
.project	4/23/2015 8:44 PM	PROJECT File	1 KB
testng	4/23/2015 8:49 PM	XML File	1 KB

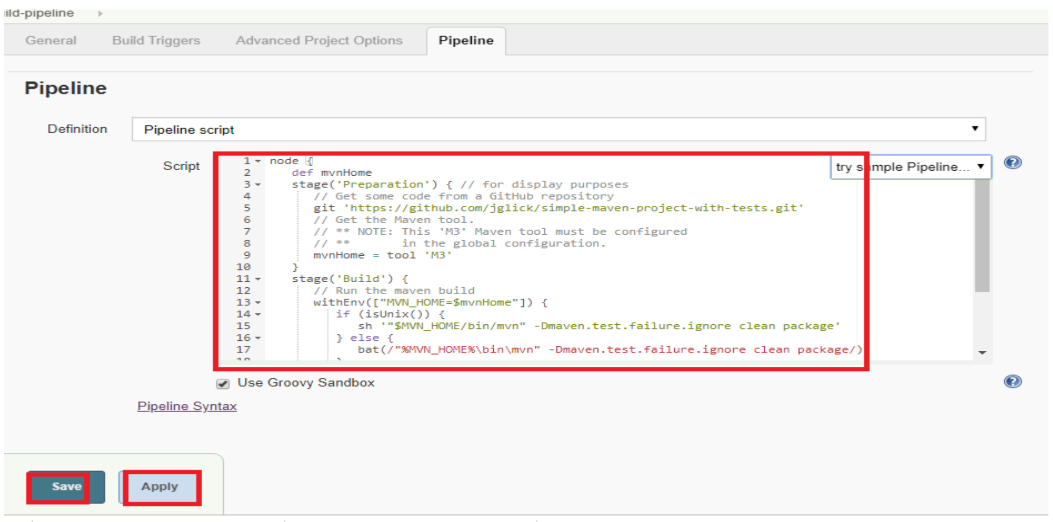
Date created: 4/23/2015 8:44 PM
Size: 212 bytes
Folders: demoJenkins

- Create a text file **run.bat** and keep it within double quotation marks.


run - Notepad

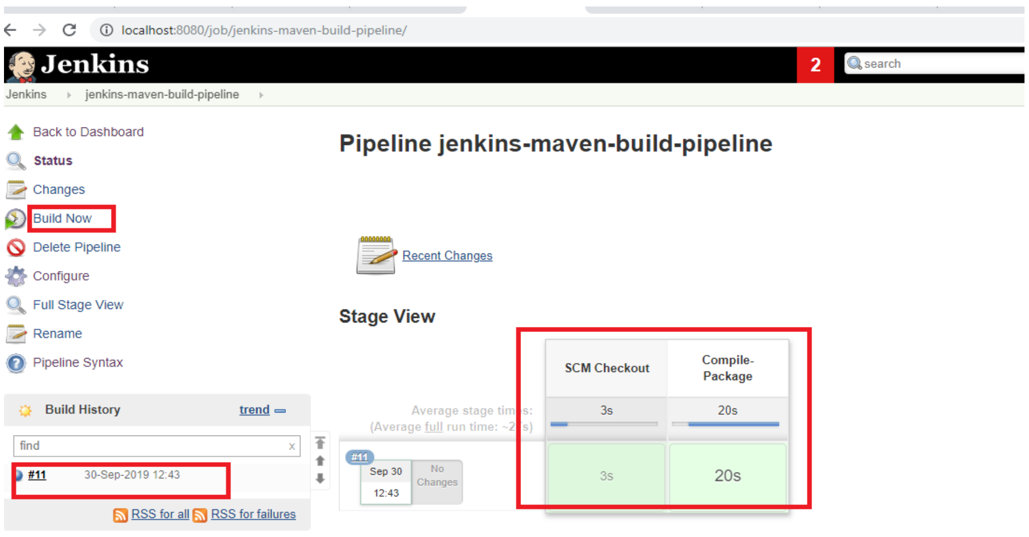
```
java -cp bin;lib/* org.testng.TestNG testng.xml
```

- Go to the Jenkins Pipeline job.
- Pass the Selenium script location.
- Write a groovy script in Pipeline.
- Click on Apply and then Save.



```
node {
    def mvnHome
    stage('Preparation') { // for display purposes
        // Get some code from a GitHub repository
        git 'https://github.com/jglick/simple-maven-project-with-tests.git'
        // Get the Maven tool.
        // ** NOTE: This 'M3' Maven tool must be configured
        // **       in the global configuration.
        mvnHome = tool 'M3'
    }
    stage('Build') {
        // Run the maven build
        withEnv(["MVN_HOME=$mvnHome"]) {
            if (isUnix()) {
                sh '"$MVN_HOME/bin/mvn" -Dmaven.test.failure.ignore clean package'
            } else {
                bat(/"%MVN_HOME%\bin\mvn" -Dmaven.test.failure.ignore clean package/)
```

- Now, click on **Build Now.**



20. TDD with TestNG

Step 20.1: Performing a TDD test:

- To perform a TDD test, follow the below steps:
    a. Add the test.

b. Execute the test and see if the new one fails.
c. Write the code.
d. Execute the test.
e. Refactor the code.
f. Now, repeat the steps mentioned above.

Now, let's look at the above steps in detail:

a. Firstly, write the code that will be based on the requirements in Eclipse. It should look something like:

```
package test.testing;

import org.testng.Assert;
import org.testng.annotations.Test;

public class AddNumbers {

	@Test

	public void addIntegerNumbers()
	{
	Calculator myCalculator = new Calculator();
	int expected= 30;
	int actual= myCalculator.add(10,20);
	Assert.assertEquals(actual, expected);
	}
}
```

b. Now, if we execute our test for the first time, we should get the below error:

```
FAILED: addIntegerNumbers
java.lang.Error: Unresolved compilation problems:
Calculator cannot be resolved to a type
Calculator cannot be resolved to a type
```

c. Write the code shown below to resolve the above error in Eclipse. It will look something like:

```
package test.testing;

public class Calculator {

public int add(int number1, int number2)

  {
      return 0;

      }

    }
```

d. Now, execute our test:

```
FAILED: addIntegerNumbers
java.lang.AssertionError: expected [30] but found [0]
```

e. Refactor the code in Eclipse. It will look like:

```
package test.testing;

public class Calculator {

public int add(int number1, int number2)
{
return (number1+number2);
}
```

f. Now, if execute our test again, it will show the below message:

```
PASSED: addIntegerNumbers
```

Step 20.2: Running the code:

- Run the code through Eclipse.