Table of Contents

[1. Setup 2](#_Toc363461745)

[1.1. Java 2](#_Toc363461746)

[1.2. IntelliJ/Eclipse 2](#_Toc363461747)

[1.3. Selenium Java Client Driver 2](#_Toc363461748)

[1.4. Chromedriver/IEdriver 3](#_Toc363461749)

[1.5. Apache Maven 3](#_Toc363461750)

[1.6. Ant 3](#_Toc363461751)

[1.7. M2eclipse 3](#_Toc363461752)

[2. Create new project 3](#_Toc363461753)

[3. Language tips 5](#_Toc363461754)

[3.1. Static 5](#_Toc363461759)

[3.2. Css selector 6](#_Toc363461760)

[3.3. Wait command 6](#_Toc363461761)

[3.3.1. Implicit Wait 6](#_Toc363461774)

[3.3.2. Explicit Wait 6](#_Toc363461775)

[3.4. Checkbox, radio button, drop down box 7](#_Toc363461776)

[3.5. Mouse over 7](#_Toc363461777)

[3.6. Building a Series of Multiple Actions 8](#_Toc363461778)

[4. Selenium RC and Selenium Webdriver 9](#_Toc363461779)

[4.1. Both WebDriver and Selenium RC have following features 9](#_Toc363461784)

[4.2. Different 10](#_Toc363461785)

[5. TestNG (Test next generation) 11](#_Toc363461786)

[5.1. Installing TestNG in Eclipse 11](#_Toc363461789)

[5.2. Setting up a new TestNG Project 14](#_Toc363461790)

[5.3. Creating a New TestNG Test File 18](#_Toc363461791)

[5.4. Coding Our First Test Case 20](#_Toc363461792)

[5.5. Running the Test 21](#_Toc363461793)

[5.6. Checking reports created by TestNG 22](#_Toc363461794)

[5.7. Generating HTML Reports 23](#_Toc363461795)

[5.8. Annotations used in TestNG 25](#_Toc363461796)

[5.8.1. Multiple Test Cases 25](#_Toc363461819)

[5.8.2. Parameters 26](#_Toc363461820)

[5.8.3. Multiple Parameters 27](#_Toc363461821)

[5.8.4. Summary of TestNG Annotations 31](#_Toc363461822)

[5.9. Conclusion 31](#_Toc363461823)

[6. Selenium Grid 31](#_Toc363461824)

[6.1. What is Selenium Grid? 31](#_Toc363461826)

[6.2. When to Use Selenium Grid 32](#_Toc363461827)

[6.3. Grid 1.0 Vs Grid 2.0 32](#_Toc363461828)

[6.4. What is a Hub and Node 32](#_Toc363461829)

[6.4.1. The Hub 32](#_Toc363461844)

[6.4.2. The Nodes 32](#_Toc363461845)

[6.5. How to Install and Use Grid 2.0 33](#_Toc363461846)

[6.6. Designing Test Scripts That can run on the grid 35](#_Toc363461847)

[6.6.1. Using the DesiredCapabilites Object 36](#_Toc363461850)

[6.6.2. Using the RemoteWebDriver Object 36](#_Toc363461851)

[6.7. Running a Sample Test Case on the Grid 36](#_Toc363461852)

[6.8. Summary 37](#_Toc363461853)

**SELENIUM WEB DRIVER**

# Setup

## Java

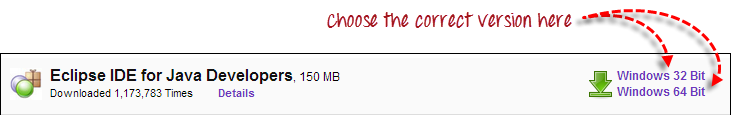


SET JAVA\_HOME=C:\Program Files\Java\jdk1.7.0\_04

set PATH=%PATH%;%JAVA\_HOME%\bin

## IntelliJ/Eclipse

Eclipse IDE for Java EE Developers (<http://www.eclipse.org/downloads/>)



## Selenium Java Client Driver



This download comes as a ZIP file named “selenium-2.25.0.zip”. For simplicity, extract the contents of this ZIP file on your C drive so that you would have the directory “C:\selenium-2.25.0\”. This directory contains all the JAR files that we would later import on Eclipse.  
Read more at <http://www.guru99.com/installing-selenium-webdriver.html#GZUHrQJqvTgEdHbE.99>

## Chromedriver/IEdriver

<https://code.google.com/p/chromedriver/downloads/list>

|  |  |  |
| --- | --- | --- |
| **Browser** | **Name of Driver Server** | **Remarks** |
| HTMLUnit | (none) | WebDriver can drive HTMLUnit without the need of a driver server |
| Firefox | (none) | WebDriver can drive Firefox without the need of a driver server |
| Internet Explorer | Internet Explorer Driver Server | Available in 32 and 64-bit versions. Use the version that corresponds to the architecture of your IE |
| Chrome | ChromeDriver | Though its name is just “ChromeDriver”, it is in fact a Driver Server, not just a driver. The current version can support versions higher than Chrome v.21 |
| Opera | OperaDriver | Though its name is just “OperaDriver”, it is in fact a Driver Server, not just a driver. |
| PhantomJS | GhostDriver | PhantomJS is another headless browser just like HTMLUnit. |
| Safari | SafariDriver | Though its name is just “SafariDriver”, it is in fact a Driver Server, not just a driver. |

Read more at <http://www.guru99.com/installing-selenium-webdriver.html#GZUHrQJqvTgEdHbE.99>

## Apache Maven

<http://maven.apache.org/download.cgi>

mvn clean install

mvn -version

set M2\_HOME= C:\Program Files\apache-maven-3.0.4

set PATH=%PATH%;%M2\_HOME%\bin

## Ant

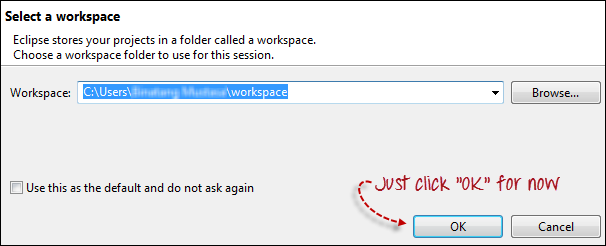
(For Selenium RC)

## M2eclipse

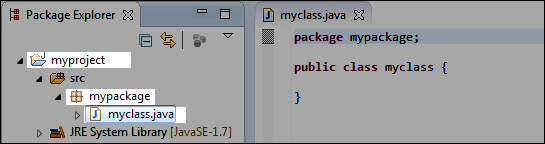
* Eclipse does not have integrated Maven support out of the box. To add the support, I am going to use [Maven Integration (m2e)](http://eclipse.org/m2e).
* In Eclipse: Help -> Install New Software…
* Type the following URL in field Work with:<http://download.eclipse.org/technology/m2e/releases>
* Click Add…
* Give a name for the repository, such as:
* m2eclipse
* Click OK
* Select the checkbox Maven Integration for Eclipse
* Click Next etc. to move forward and choose to restart Eclipse when prompted

# Create new project

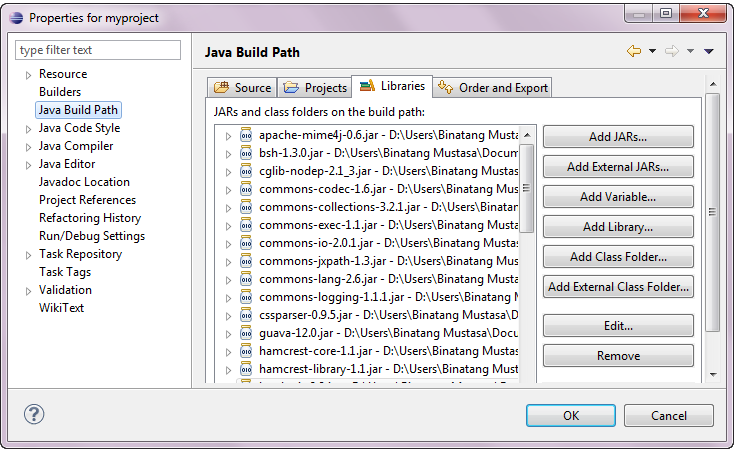
* Launch the “eclipse.exe” file inside the “eclipse” folder that we extracted in step 2. If you followed step 2 correctly, the executable should be located on C:\eclipse\eclipse.exe.
* When asked to select for a workspace, just accept the default location.



* Create a new project through File > New > Java Project. Name the project as “myproject”.
* Right-click on the newly created project and select New > Package, and name that package as “mypackage”.
* Create a new Java class under mypackage by right-clicking on it and then selecting New > Class, and then name it as “myclass”. Your Eclipse IDE should look like the image below.



* Right-click on myproject and select Properties.
* On the Properties dialog, click on “Java Build Path”.
* Click on the Libraries tab, and then click “Add External JARs..”
* Navigate to C:\selenium-2.25.0\ (or any other location where you saved the extracted contents of “selenium-2.25.0.zip” in step 3).
* Add all the JAR files inside and outside the “libs” folder. Your Properties dialog should now look similar to the image below.



* Finally, click OK and we are done importing Selenium libraries into our project.

Read more at <http://www.guru99.com/installing-selenium-webdriver.html#qApkB11AZKEf8QMv.99>

# Language tips



## Static

* Khi bạn new nhiều đối tượng A thì biến static trong các đối tượng A có giá trị giống nhau. Trong khi biến non static bị trả về 0 mỗi khi new đối tượng mới.

public class StaticSample {

public static int count = 0;

public int a = 0;

public StaticSample() {

count++;

a++;

}

public static void main(String[] args) {

StaticSample aClass = new StaticSample();

StaticSample bClass = new StaticSample();

//Đếm số đối tượng đã được tạo (biến static ko bị gán lại)

System.out.println(StaticSample.count);

//Các Biến non static bị gán lại ứng với mỗi đối tượng

System.out.println(aClass.getA());

System.out.println(bClass.getA());

}

public int getA() {

return a;

}

}

Kết quả :  
2  
1  
1

* Hàm static được sử dụng khi cần có một lớp tiện ích dùng chung (Util) trong chương trình. Việc tạo một lớp Util rồi trực tiếp tham chiếu đến hàm cần dùng thông qua class làm cho thiết kế tốt hơn so với việc phải khởi tạo một đối tượng rồi dùng hàm tham chiếu từ đối tượng đó. Vì đôi khi việc khởi tạo đối tượng lại tốn chi phí lớn, ảnh hưởng tới performance.
* Nội trong 1 class:
  + 1 static method: chỉ có thể gọi được các static method, property khác trong class đó. Không thể gọi được 1 non-static method, property
  + 1 non-static method: có thể gọi bất kỳ static hay non-static method, property trong class đó
* Class ngoài:
  + Gọi thông qua Class: chỉ có thể gọi được các static method, property trong Class được gọi
  + Khởi tạo đối tượng (Object) từ Class rồi gọi: có thể gọi bất kỳ static và non-static

## Css selector

<http://www.w3schools.com/cssref/css_selectors.asp>

## Wait command

There are two kinds of waits.

* Implicit wait – used to set the default waiting time throughout the program
* Explicit wait – used to set the waiting time for a particular instance only



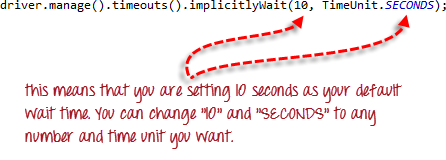
### Implicit Wait

* It is simpler to code than Explicit Waits.
* It is usually declared in the instantiation part of the code.
* You will only need one additional package to import.

To start using an implicit wait, you would have to import this package into your code.

http://www.guru99.com/images/image038(1).png

Then on the instantiation part of your code, add this.



### Explicit Wait

**Explicit waits are done using the WebDriverWait and ExpectedCondition classes**. For the following example, we shall wait up to 10 seconds for an element whose id is “username” to become visible before proceeding to the next command. Here are the steps.

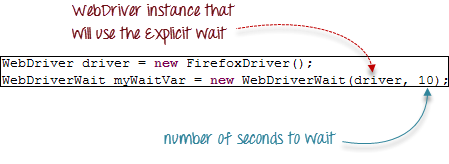
**Step 1**

Import these two packages:

**http://www.guru99.com/images/image040(1).png**

**Step 2**

Declare a WebDriverWait variable. In this example, we will use “myWaitVar” as the name of the variable.



**Step 3**

Use myWaitVar with ExpectedConditions on portions where you need the explicit wait to occur. In this case, we will use explicit wait on the “username” (Mercury Tours HomePage) input before we type the text “tutorial” onto it.

http://www.guru99.com/images/image042(1).png

## Checkbox, radio button, drop down box

* The table below summarizes the commands to access each type of element discussed above.

|  |  |  |
| --- | --- | --- |
| **Element** | **Command** | **Description** |
| **Input Box** | sendKeys() | used to enter values onto text boxes |
| clear() | used to clear text boxes of its current value |
| **Check Box,**  **Radio Button,** | click() | used to toggle the element on/off |
| **Links** | click() | used to click on the link and wait for page load to complete before proceeding to the next command. |
| **Drop-Down Box** | selectByVisibleText()/  deselectByVisibleText() | selects/deselects an option by its displayed text |
| selectByValue()/  deselectByValue() | selects/deselects an option by the value of its “value” attribute |
| selectByIndex()/  deselectByIndex() | selects/deselects an option by its index |
| isMultiple() | returns TRUE if the drop-down element allows multiple selection at a time; FALSE if otherwise |
| deselectAll() | deselects all previously selected options |
| **Submit Button** | submit() |  |

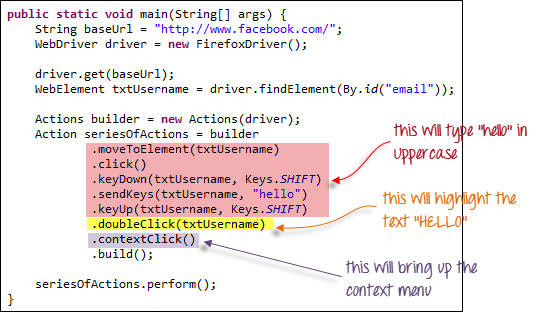
* WebDriver allows selection of more than one option in a multiple SELECT element.
* To control drop-down boxes, you must first import the org.openqa.selenium.support.ui.Select package and then create a Select instance.
* You can use the submit() method on any element within the form. WebDriver will automatically trigger the submit function of the form where that element belongs to

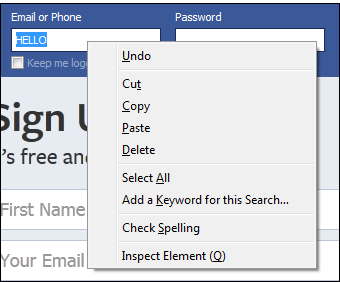
## Mouse over

|  |  |
| --- | --- |
| **Method** | **Description** |
| **clickAndHold()** | Clicks (without releasing) at the current mouse location. |
| **contextClick()** | Performs a context-click at the current mouse location. |
| **doubleClick()** | Performs a double-click at the current mouse location. |
| **dragAndDrop(source, target)** | Performs click-and-hold at the location of the source element, moves to the location of the target element, then releases the mouse.  **Parameters:**  source- element to emulate button down at.  target- element to move to and release the mouse at. |
| **dragAndDropBy(source, x-offset, y-offset)** | Performs click-and-hold at the location of the source element, moves by a given offset, then releases the mouse.  **Parameters**:  source- element to emulate button down at.  xOffset- horizontal move offset.  yOffset- vertical move offset. |
| **keyDown(modifier\_key)** | Performs a modifier key press. Does not release the modifier key - subsequent interactions may assume it's kept pressed.  **Parameters**:  modifier\_key – any of the modifier keys (Keys.ALT, Keys.SHIFT, or Keys.CONTROL) |
| **keyUp(modifier \_key)** | Performs a key release.  **Parameters**:  modifier\_key – any of the modifier keys (Keys.ALT, Keys.SHIFT, or Keys.CONTROL) |
| **moveByOffset(x-offset, y-offset)** | Moves the mouse from its current position (or 0,0) by the given offset.  **Parameters**:  x-offset- horizontal offset. A negative value means moving the mouse left.  y-offset- vertical offset. A negative value means moving the mouse up. |
| **moveToElement(toElement)** | Moves the mouse to the middle of the element. **Parameters**:  toElement- element to move to. |
| **release()** | Releases the depressed left mouse button at the current mouse location |
| **sendKeys(onElement, charsequence)** | Sends a series of keystrokes onto the element. **Parameters**:  onElement- element that will receive the keystrokes, usually a text field  charsequence– any string value representing the sequence of keystrokes to be s |

## Building a Series of Multiple Actions

**You can build a series of actions using the Action and Actions classes**. Just remember to close the series with the build() method. Consider the sample code below.





# Selenium RC and Selenium Webdriver



## Both WebDriver and Selenium RC have following features

* They both allow you to use a programming language in designing your test scripts.
* They both allow you to run your tests against different browsers.

## Different

|  |  |
| --- | --- |
| http://newguru99.revolutionventur.netdna-cdn.com/images/2-way_handshake.jpg | http://newguru99.revolutionventur.netdna-cdn.com/images/3-way_handshake_2.jpg |
| http://newguru99.revolutionventur.netdna-cdn.com/images/simplified_rc_architecture.jpg | http://newguru99.revolutionventur.netdna-cdn.com/images/simplified_webdriver_architecture.jpg |

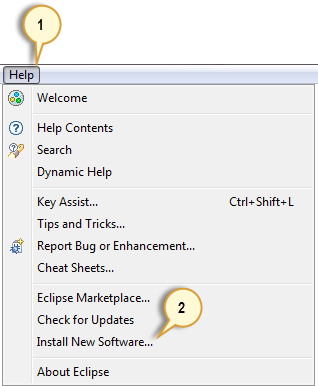
# TestNG (Test next generation)



## Installing TestNG in Eclipse

**Step 1**

* Launch Eclipse.
* On the menu bar, click Help.
* Choose the “Install New Software…” option.



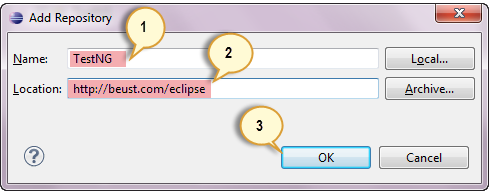
**Step 2**

In the Install dialog box, click the Add button

http://newguru99.revolutionventur.netdna-cdn.com/images/step_2-0047.png

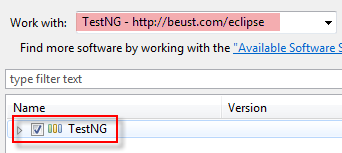
**Step 3**

1. In “Name”, type TestNG.
2. In “Location”, type <http://beust.com/eclipse>.
3. Click OK



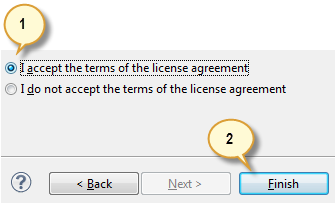
**Step 4**

* Notice that “TestNG - <http://beust.com/eclipse>” was populated onto the “Work with:” textbox.
* Check the “TestNG” check box as shown below, then click Next.



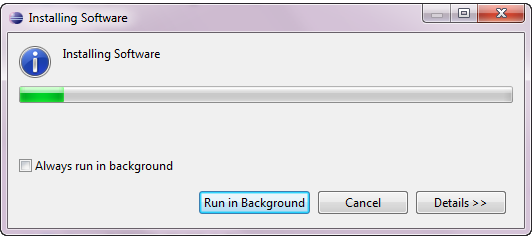
**Step 5**

* Click Next again on the succeeding dialog box until you reach the License Agreement dialog.
* Click “I accept the terms of the license agreement” then click Finish.

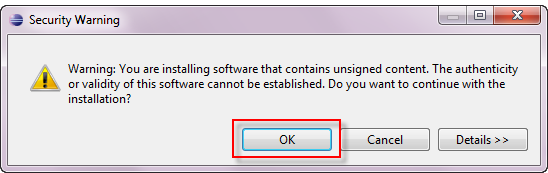


**Step 6**

Wait for the installation to finish

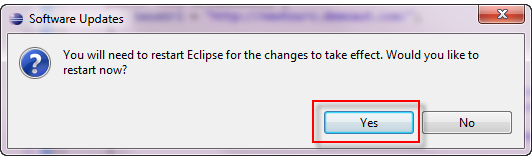


If you encounter a Security warning, just click OK



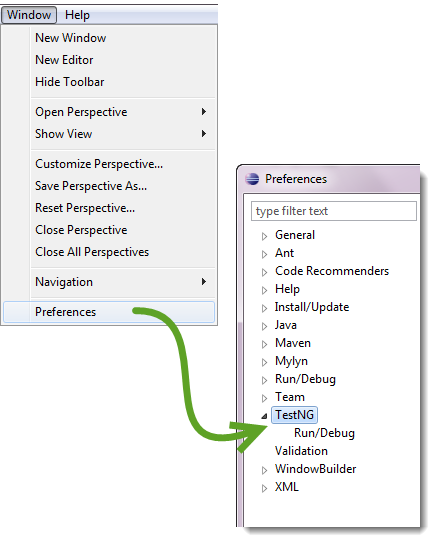
**Step 7**

When Eclipse prompts you for a restart, just click Yes.



**Step 8**

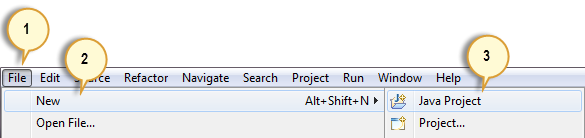
After restart, verify if TestNG was indeed successfully installed. Click Window > Preferences and see if TestNG is included on the Preferences list.



## Setting up a new TestNG Project

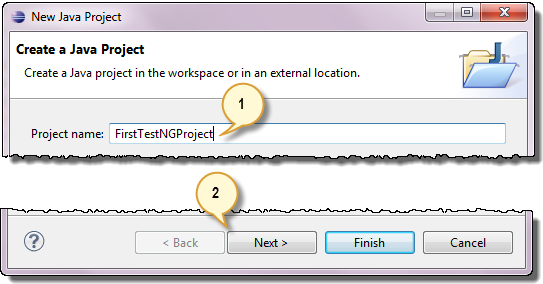
**Step 1**

Click File > New > Java Project



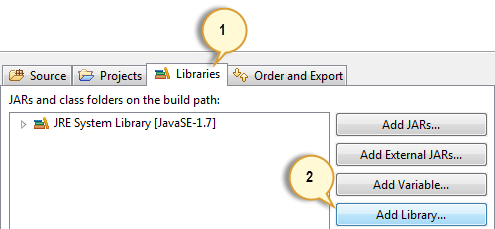
**Step 2**

Type “FirstTestNGProject” as the Project Name then click Next.



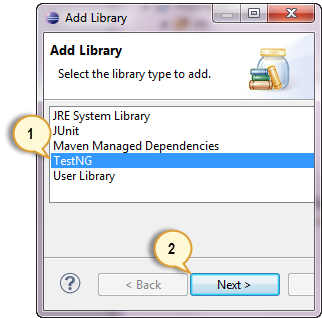
**Step 3**

We will now start to import the TestNG Libraries onto our project. Click on the “Libraries” tab, and then “Add Library…”



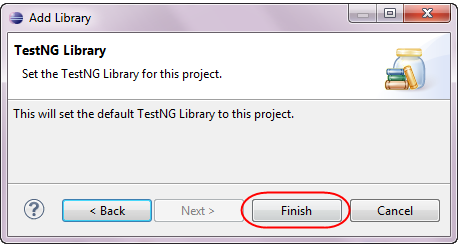
**Step 4**

On the Add Library dialog, choose “TestNG” and click Next.

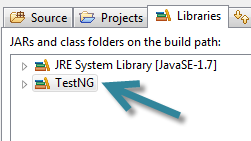


**Step 5**

Click Finish.

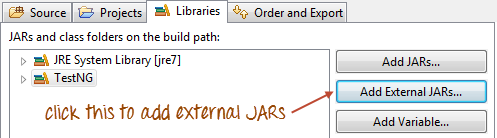


You should notice that TestNG is included on the Libraries list.

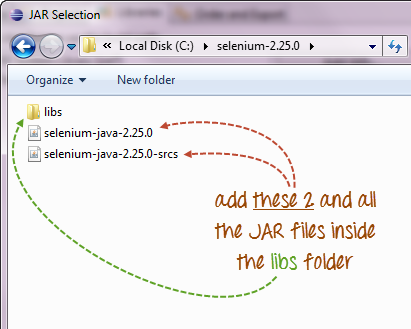


**Step 6**

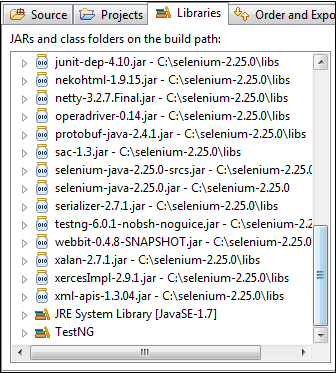
We will now add the JAR files that contain the Selenium API. These files are found in the Java client driver that we downloaded from <http://docs.seleniumhq.org/download/> when we were installing Selenium and Eclipse in the previous chapters.



Then, navigate to where you have placed the Selenium JAR files.

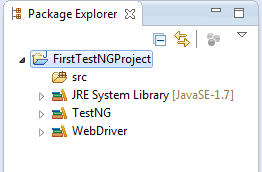


After adding the external JARs, your screen should look like this.



**Step 7**

Click Finish and verify that our FirstTestNGProject is visible on Eclipse’s Package Explorer window.



## Creating a New TestNG Test File

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

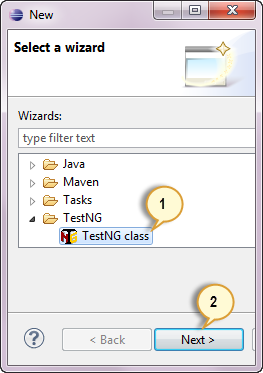
**Step 1**

Right-click on the “src” package folder then choose New > Other…



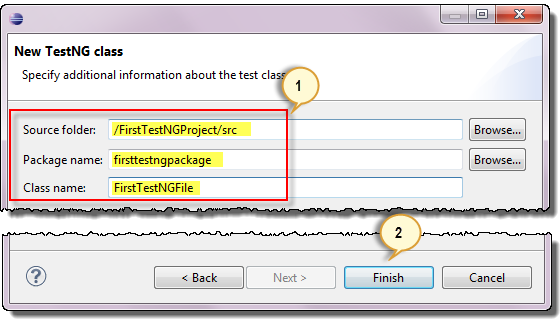
**Step 2**

Click on the TestNG folder and select the “TestNG class” option. Click Next.

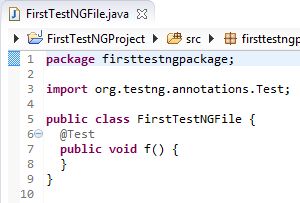


**Step 3**

Type the values indicated below on the appropriate input boxes and click Finish. Notice that we have named our Java file as “FirstTestNGFile”.

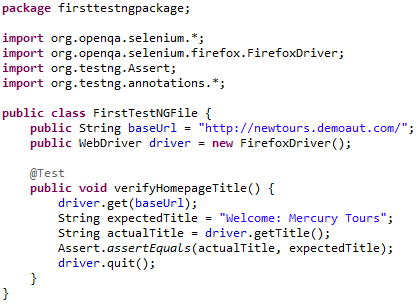


Eclipse should automatically create the template for our TestNG file shown below.



## Coding Our First Test Case

Let us now create our first test case that will check if Mercury Tours’ homepage is correct. Type your code as shown below.



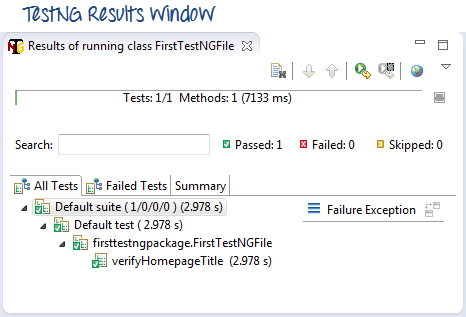
Notice the following.

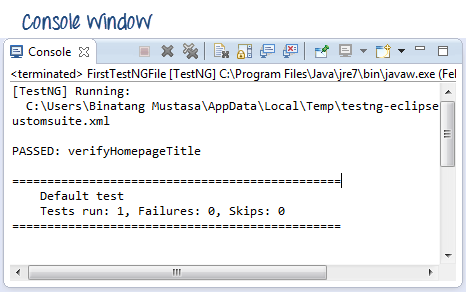
* TestNG does not require you to have a main() method.
* Methods need not be static.
* We used the @Test annotation. **@Test is used to tell that the method under it is a test case**. In this case, we have set the verifyHomepageTitle() method to be our test case so we placed an ‘@Test’ annotation above it.
* Since we use annotations in TestNG, we needed to import the package org.testng.annotations.\*.
* We used the Assert class. **The Assert class is used to conduct verification operations in TestNG**. To use it, we need to import the org.testng.Assert package.

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

## Running the Test

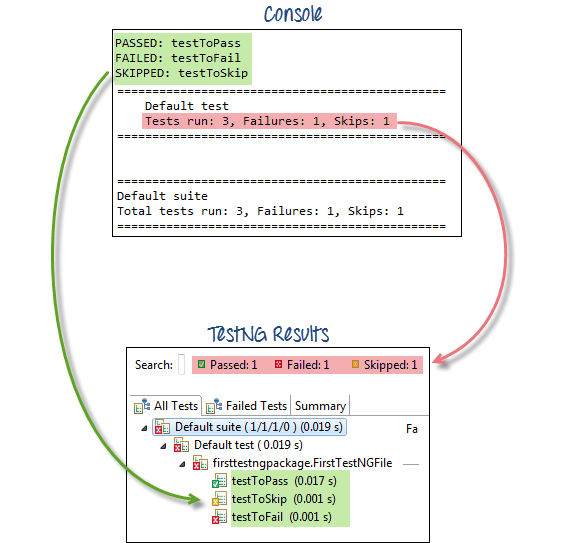
To run the test, simply run the file in Eclipse as you normally do. Eclipse will provide two outputs – one in the Console window and the other on the TestNG Results window.





## Checking reports created by TestNG

The Console window in Eclipse gives a text-based report of our test case results while the TestNG Results window gives us a graphical one.

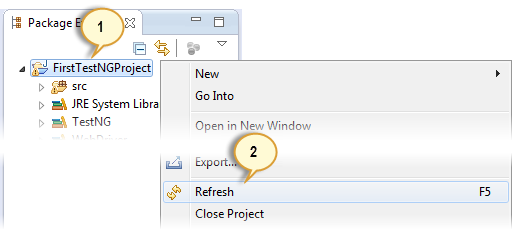


## Generating HTML Reports

TestNG has the ability to generate reports in HTML format.

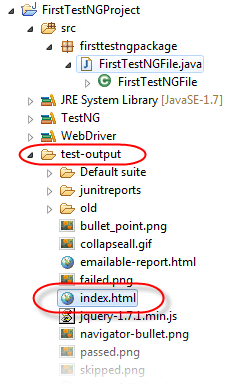
**Step 1**

After running our FirstTestNGFile that we created in the previous section, right-click the project name (FirstTestNGProject) in the Project Explorer window then click on the “Refresh” option.



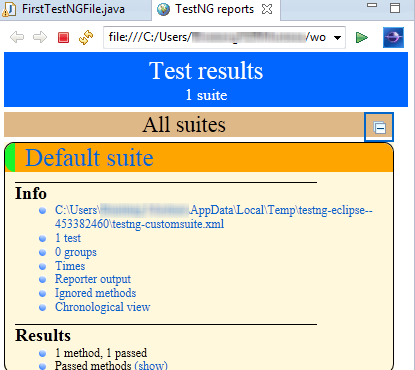
**Step 2**

Notice that a “test-output” folder was created. Expand it and look for an index.html file. This HTML file is a report of the results of the most recent test run.



**Step 3**

Double-click on that index.html file to open it within Eclipse’s built-in web browser. You can refresh this page any time after you rerun your test by simply pressing F5 just like in ordinary web browsers.



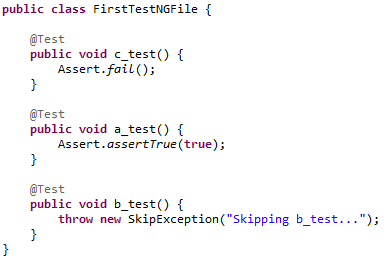
## Annotations used in TestNG

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

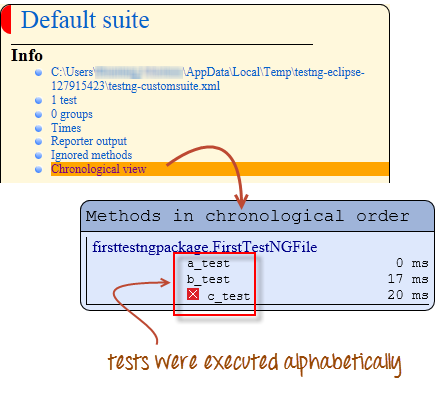


### Multiple Test Cases

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



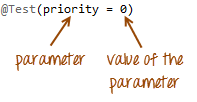
Run this code and on the generated index.html page, click “Chronological view”.



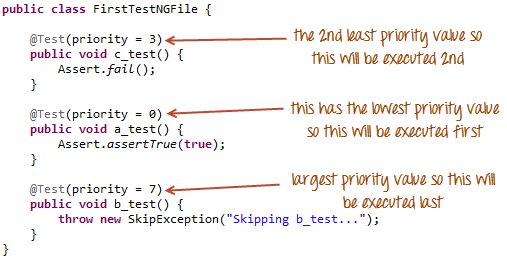
### Parameters

If you want the methods to be executed in a different order, use the parameter “priority”. **Parameters are keywords that modify the annotation’s function**.

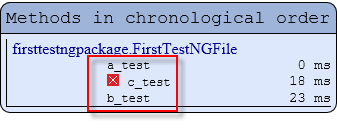
* Parameters require you to assign a value to them. You do.this by placing a “=” next to them, and then followed by the value.
* Parameters are enclosed in a pair of parentheses which are placed right after the annotation like the code snippet shown below.



TestNG will execute the @Test annotation with the lowest priority value up to the largest. There is no need for your priority values to be consecutive.



The TestNG HTML report will confirm that the methods were executed based on the ascending value of priority.



### Multiple Parameters

Aside from “priority”, @Test has another parameter called “alwaysRun” which can only be set to either “true” or “false”. **To use two or more parameters in a single annotation, separate them with a comma** such as the one shown below.

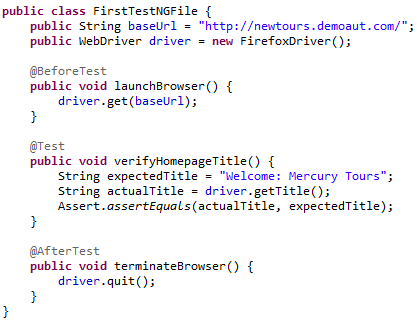
http://newguru99.revolutionventur.netdna-cdn.com/images/multiple_parameters-0010.png



@BeforeTest and @AfterTest

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

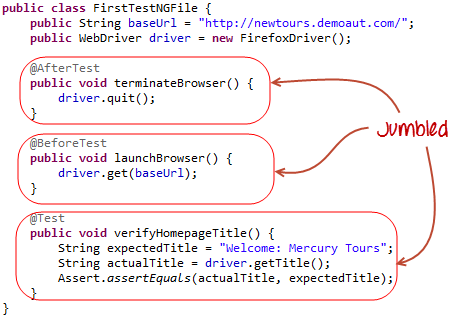
Consider the code below.



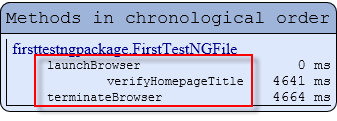
Applying the logic presented by the table and the code above, we can predict that the sequence by which methods will be executed is:

* 1st – launchBrowser()
* 2nd – verifyHomepageTitle()
* 3rd – terminateBrowser()

**The placement of the annotation blocks can be interchanged without affecting the chronological order by which they will be executed**. For example, try to rearrange the annotation blocks such that your code would look similar to the one below.



Run the code above and notice that



@BeforeMethod and @AfterMethod

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

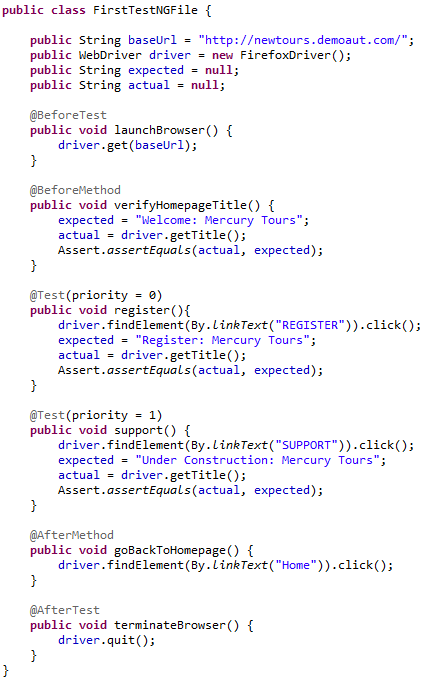
In Mercury Tours, suppose we like to verify the titles of the target pages of the two links below.



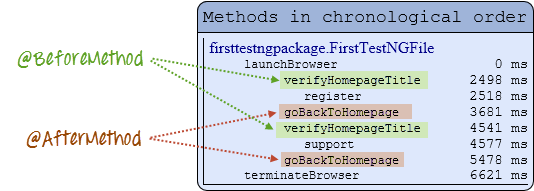
The flow of our test would be:

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

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



After executing this test, your TestNG should report the following sequence.



Simply put, @BeforeMethod should contain methods that you need to run **before** each test case while @AfterMethod should contain methods that you need to run **after** each test case.

### Summary of TestNG Annotations

**@BeforeSuite**: The annotated method will be run before all tests in this suite have run.

**@AfterSuite**: The annotated method will be run after all tests in this suite have run.

**@BeforeTest**: The annotated method will be run before any test method belonging to the classes inside the tag is run.

**@AfterTest**: The annotated method will be run after all the test methods belonging to the classes inside the tag have run.

**@BeforeGroups**: The list of groups that this configuration method will run before. This method is guaranteed to run shortly before the first test method that belongs to any of these groups is invoked.

**@AfterGroups**: The list of groups that this configuration method will run after. This method is guaranteed to run shortly after the last test method that belongs to any of these groups is invoked.

**@BeforeClass**: The annotated method will be run before the first test method in the current class is invoked.

**@AfterClass**: The annotated method will be run after all the test methods in the current class have been run.

**@BeforeMethod**: The annotated method will be run before each test method.

**@AfterMethod**: The annotated method will be run after each test method.

**@Test**: The annotated method is a part of a test case

## Conclusion

* TestNG is a testing framework that is capable of making Selenium tests easier to understand and of generating reports that are easy to understand.
* The main advantages of TestNG over JUnit are the following.
* Annotations are easier to use and understand.
* Test cases can be grouped more easily.
* TestNG allows us to create parallel tests.
* The Console window in Eclipse generates a text-based result while the TestNG window is more useful because it gives us a graphical output of the test result plus other meaningful details such as:
  + Runtimes of each method.
  + The chronological order by which methods were executed.
* TestNG is capable of generating HTML-based reports.
* Annotations can use parameters just like the usual Java methods.

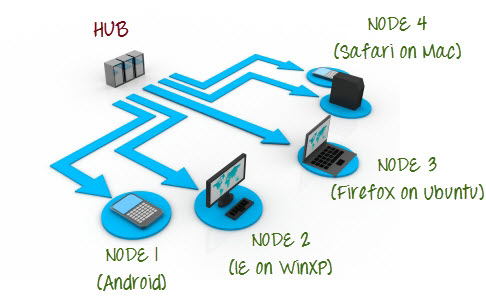
# Selenium Grid



## What is Selenium Grid?

**Selenium Grid is a part of the Selenium Suite that specializes on running multiple tests across different browsers, operating systems, and machines in parallel**.

Selenium Grid has 2 versions – the older Grid 1 and the newer Grid 2. We will only focus on Grid 2 because Grid 1 is gradually being deprecated by the Selenium Team.



Selenium Grid uses a hub-node concept where you only run the test on a single machine called a **hub**, but the execution will be done by different machines called **nodes**.

## When to Use Selenium Grid

You should use Selenium Grid when you want to do either one or both of following :

* **Run your tests against different browsers, operating systems, and machines all at the same time.**This will ensure that the application you are testing is fully compatible with a wide range of browser-OS combinations.
* **Save time in execution of your test suites**. If you set up Selenium Grid to run, say, 4 tests at a time, then you would be able to finish the whole suite around 4 times faster.

## Grid 1.0 Vs Grid 2.0

Following are the main differences between Selenium Grid 1 and 2.

|  |  |
| --- | --- |
| **Grid 1** | **Grid 2** |
| Selenium Grid 1 has its own remote control that is different from the Selenium RC server. They are two different programs. | Selenium Grid 2 is now bundled with the Selenium Server jar file |
| You need to install and configure Apache Ant first before you can use Grid 1. | You do not need to install Apache Ant in Grid 2. |
| Can only support Selenium RC commands/scripts. | Can support both Selenium RC and WebDriver scripts. |
| You can only automate one browser per remote control. | One remote control can automate up to 5 browsers. |

## What is a Hub and Node



### The Hub

* The hub is the central point where you load your tests into.
* There should only be one hub in a grid.
* The hub is launched only on a single machine, say, a computer whose OS is Windows 7 and whose browser is IE.
* The machine containing the hub is where the tests will be run, but you will see the browser being automated on the node.

### The Nodes

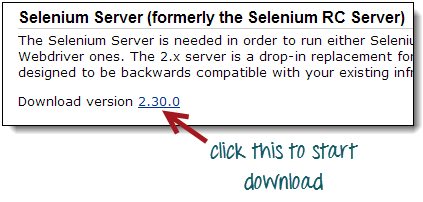
* Nodes are the Selenium instances that will execute the tests that you loaded on the hub.
* There can be one or more nodes in a grid.
* Nodes can be launched on multiple machines with different platforms and browsers.
* The machines running the nodes need not be the same platform as that of the hub.

## How to Install and Use Grid 2.0

In this section, you will use 2 machines. The first machine will be the system that will run the hub, while the other machine will run a node. For simplicity, let us call the machine where the hub runs as “Machine A” while the machine where the node runs will be “Machine B”. It is also important to note their IP addresses. Let us say that Machine A has an IP address of 192.168.1.3 while Machine B has an IP of 192.168.1.4.

**Step 1**

Download the Selenium Server by [here](http://docs.seleniumhq.org/download/).



**Step 2**

You can place the Selenium Server .jar file anywhere in your HardDrive.But for the purpose of this tutorial, place it on the C drive of both Machine A and Machine B. After doing this, you are now done installing Selenium Grid. The following steps will launch the hub and the node.

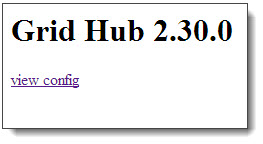
**Step 3**

* We are now going to launch a hub. Go to Machine A. Using the command prompt, navigate to the root of Machine A’s - C drive ,because that is the directory where we placed the Selenium Server.
* On the command prompt, type **java –jar selenium-server-standalone-2.30.0.jar –role hub**
* The hub should successfully be launched. Your command prompt should look similar to the image below



**Step 4**

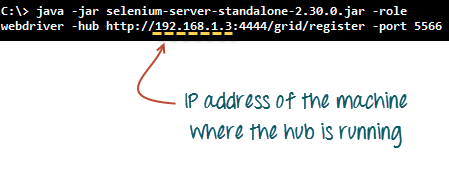
Another way to verify whether the hub is running is by using a browser. Selenium Grid, by default, uses Machine A’s port 4444 for its web interface. Simply open up a browser and go to <http://localhost:4444/grid/console>



Also, you can check if Machine B can access the hub’s web interface by launching a browser there and going to [http://iporhostnameofmachineA:4444/grid/console](http://iporhostnameofmachinea:4444/grid/console) where “iporhostnameofmachineA” should be the IP address or the hostname of the machine where the hub is running. Since Machine A’s IP address is 192.168.1.3, then on the browser on Machine B you should type [http://192.168.1.3:4444/grid/console](http://192.168.1.11:4444/grid/console)

**Step 5**

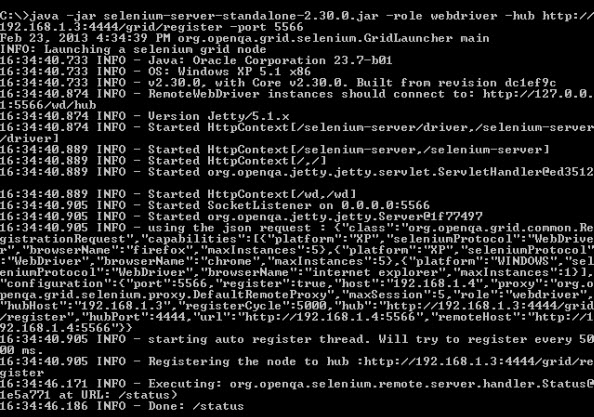
* Now that the hub is already set up, we are going to launch a node. Go to Machine B and launch a command prompt there.
* Navigate to the root of Drive C and type the code below. We used the IP address 192.168.1.3 because that is where the hub is running. We also used port 5566 though you may choose any free port number you desire.



* If you want to run IE:

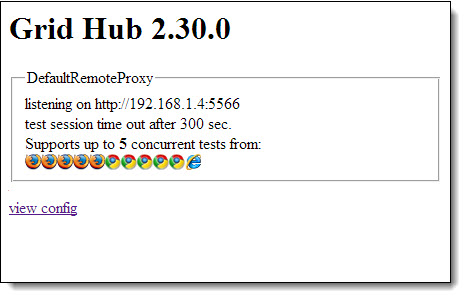
java -jar selenium-server-standalone-2.26.0.jar -Dwebdriver.ie.driver=C:\Temp\IEDriverServer.exe -rol webdriver -hub http://192.168.1.3:4444/grid/register -port 5566

* When you press Enter, your command prompt should be similar to the image below.



**Step 6**

Go to the Selenium Grid web interface and refresh the page. You should see something like this.



At this point, you have already configured a simple grid. You are now ready to run a test remotely on Machine B.

## Designing Test Scripts That can run on the grid

To design test scripts that will run on the grid, we need to use **DesiredCapabilites** and the **RemoteWebDriver** objects.

* **DesiredCapabilites**is used to set the type of **browser** and **OS**that we will automate
* **RemoteWebDriver**is used to set which node (or machine) that our test will run against.

To use the **DesiredCapabilites**object, you must first import this package

http://newguru99.revolutionventur.netdna-cdn.com/images/import_desired_capabilites.png

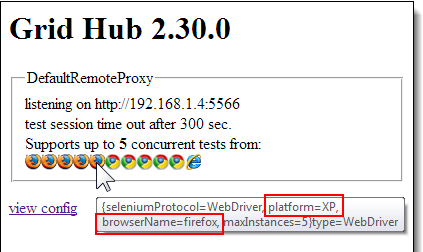
To use the **RemoteWebDriver**object, you must import these packages.

http://newguru99.revolutionventur.netdna-cdn.com/images/import_RemoteWebDriver.png



### Using the DesiredCapabilites Object

Go to the Grid’s web interface and hover on an image of the browser that you want to automate. Take note of the **platform** and the **browserName**shown by the tooltip.



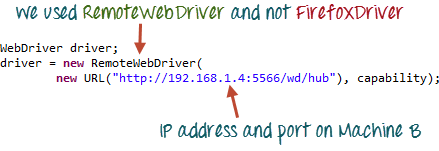
In this case, the platform is “XP” and the browserName is “firefox”.

We will use the platform and the browserName in our WebDriver as shown below (of course you need to import the necessary packages first).

http://newguru99.revolutionventur.netdna-cdn.com/images/DesiredCapabilities_code.png

### Using the RemoteWebDriver Object

Import the necessary packages for RemoteWebDriver and then pass the DesiredCapabilities object that we created above as a parameter for the RemoteWebDriver object.



## Running a Sample Test Case on the Grid

Below is a simple WebDriver TestNG code that you can create in Eclipse on Machine A. Once you run it, automation will be performed on Machine B.



 The test should pass.

|  |  |
| --- | --- |
| http://newguru99.revolutionventur.netdna-cdn.com/images/result_1.png | http://newguru99.revolutionventur.netdna-cdn.com/images/result_2.png |

## Running many node at the same time

### Create config file .xml

<suite name=*"Suite1"* verbose=*"1"* thread-count=*"2"* parallel=*"tests"*>

<tests>

<test name=*"Fire fox"* >

<parameters>

<parameter name=*"browser"* value=*"FF"*></parameter>

<parameter name=*"nodeUrl"* value=*"http://192.168.18.214:5577/wd/hub"*></parameter>

</parameters>

<classes>

<class name=*"SampleTestNG.SampleTestNG"* />

</classes>

</test>

<test name=*"IE"* >

<parameters>

<parameter name=*"browser"* value=*"IE"*></parameter>

<parameter name=*"nodeUrl"* value=*"http://192.168.37.128:5566/wd/hub"*></parameter>

</parameters>

<classes>

<class name=*"SampleTestNG.SampleTestNG"* />

</classes>

</test>

</tests>

</suite>

### Write code

**package** SampleTestNG;

**import** java.net.MalformedURLException;

**import** java.net.URL;

**import** java.util.concurrent.TimeUnit;

**import** org.openqa.selenium.By;

**import** org.openqa.selenium.Platform;

**import** org.openqa.selenium.WebDriver;

**import** org.openqa.selenium.WebElement;

**import** org.openqa.selenium.interactions.Actions;

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

**import** org.openqa.selenium.remote.RemoteWebDriver;

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

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

**import** org.testng.annotations.Parameters;

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

**public** **class** SampleTestNG{

**public** WebDriver driver;

String baseUrl;

@Parameters({"browser","nodeUrl"})

@BeforeTest(alwaysRun=**true**)

**public** **void** setUp(String browser, String nodeUrl) **throws** MalformedURLException{

baseUrl = "http://www.imdb.com/";

DesiredCapabilities capability = **new** DesiredCapabilities();

//set platform

capability.setPlatform(Platform.*ANY*);

//set browser

**if**(browser.equalsIgnoreCase("FF"))

capability=DesiredCapabilities.*firefox*();

**if**(browser.equalsIgnoreCase("IE"))

capability=DesiredCapabilities.*internetExplorer*();

//set driver

driver = **new** RemoteWebDriver(**new** URL(nodeUrl),capability);

driver.get(baseUrl);

}

@Test(priority = 0)

**public** **void** doLogin(){

//wait element

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

//mouse over to checkbox list

Actions actions = **new** Actions(driver);

WebElement menuHoverLink = driver.findElement(By.*id*("nblogin"));

actions.moveToElement(menuHoverLink).build().perform();

//login

driver.findElement(By.*xpath*("//\*[@id='nblogin']")).click();

driver.switchTo().frame(driver.findElement(By.*xpath*("/html/body/div[3]/div/div[2]/div[2]/div/iframe")));

driver.findElement(By.*id*("imdb-toggle")).click();

WebElement txtbox\_username = driver.findElement(By.*id*("usernameprompt"));

WebElement txtbox\_password = driver.findElement(By.*id*("passwordprompt"));

**if** (txtbox\_username.isEnabled()&&txtbox\_password.isEnabled()){

txtbox\_username.sendKeys("dangphuong298510@gmail.com");

txtbox\_password.sendKeys("12345678");

//dang nhap

// driver.findElement(By.className("linkasbutton-primary")).click();

driver.findElement(By.*id*("passwordprompt")).submit();

}

}

@Test(priority = 1)

**public** **void** doLogout(){

//wait element

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

//mouse over to checkbox list

Actions actions = **new** Actions(driver);

WebElement menuHoverLink = driver.findElement(By.*id*("nbpersonalize"));

actions.moveToElement(menuHoverLink).build().perform();

driver.findElement(By.*id*("nblogout")).click();

//verify logout successfully or not

**if**(driver.findElement(By.*id*("header")).getText().contains("You have logged out of IMDb")){

//if(driver.getPageSource().contains("You have logged out of IMDb")){

System.*out*.println("Logout successfully");

}**else**{

System.*out*.println("Logout unsuccessfully");

}

}

@AfterTest(alwaysRun=**true**)

**void** closeWeb(){

//close one window that WebDriver is currently controlling.

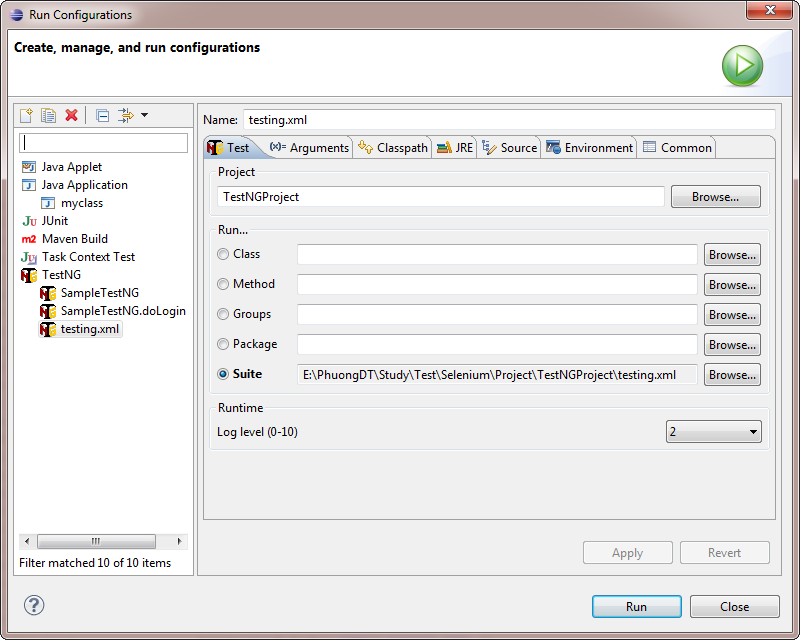
driver.close();

System.*exit*(0);

}

}

### Run config



## Summary

* Selenium Grid is used to run multiple tests simultaneously in different browsers and platforms.
* Grid uses the hub-node concept.
* The hub is the central point wherein you load your tests.
* Nodes are the Selenium instances that will execute the tests that you loaded on the hub.
* To install Selenium Grid, you only need to download the Selenium Server jar file – the same file used in running Selenium RC tests.
* There are 2 ways to verify if the hub is running: one was through the command prompt, and the other was through a browser
* To run test scripts on the Grid, you should use the DesiredCapabilities and the RemoteWebDriver objects.
* DesiredCapabilites is used to set the type of browser and OS that we will automate
* RemoteWebDriver is used to set which node (or machine) that our test will run against.

# Jenkins

## What is Jenkins?

Jenkins is an [award-winning](https://wiki.jenkins-ci.org/display/JENKINS/Awards) application that monitors executions of repeated jobs, such as building a software project or jobs run by cron. Among those things, current Jenkins focuses on the following two jobs:

* **Building/testing software projects continuously**, just like CruiseControl or DamageControl. In a nutshell, Jenkins provides an easy-to-use so-called continuous integration system, making it easier for developers to integrate changes to the project, and making it easier for users to obtain a fresh build. The automated, continuous build increases the productivity.
* **Monitoring executions of externally-run jobs**, such as cron jobs and procmail jobs, even those that are run on a remote machine. For example, with cron, all you receive is regular e-mails that capture the output, and it is up to you to look at them diligently and notice when it broke. Jenkins keeps those outputs and makes it easy for you to notice when something is wrong.

## Who is using it?

[A lot of companies and organizations](https://wiki.jenkins-ci.org/pages/viewpage.action?pageId=58001258). Most instances tend to run inside the firewall, but Google can tell you [publicly visible instances](http://www.google.com/search?ie=UTF-8&q=%22Dashboard+%5BJenkins%5D%22). We also have some statistics collected from the anonymous usage survey [here](http://jenkins-ci.org/node/282)

## Features

Jenkins offers the following features:

* **Easy installation**: Just java -jar jenkins.war, or deploy it in a servlet container. No additional install, no database.
* **Easy configuration**: Jenkins can be configured entirely from its friendly web GUI with extensive on-the-fly error checks and inline help. There's no need to tweak XML manually anymore, although if you'd like to do so, you can do that, too.
* **Change set support**: Jenkins can generate a list of changes made into the build from Subversion/CVS. This is also done in a fairly efficient fashion, to reduce the load on the repository.
* **Permanent links**: Jenkins gives you clean readable URLs for most of its pages, including some permalinks like "latest build"/"latest successful build", so that they can be easily linked from elsewhere.
* **RSS/E-mail/IM Integration**: Monitor build results by RSS or e-mail to get real-time notifications on failures.
* **After-the-fact tagging**: Builds can be tagged long after builds are completed.
* **JUnit/TestNG test reporting**: JUnit test reports can be tabulated, summarized, and displayed with history information, such as when it started breaking, etc. History trend is plotted into a graph.
* **Distributed builds**: Jenkins can distribute build/test loads to multiple computers. This lets you get the most out of those idle workstations sitting beneath developers' desks.
* **File fingerprinting**: Jenkins can keep track of which build produced which jars, and which build is using which version of jars, and so on. This works even for jars that are produced outside Jenkins, and is ideal for projects to track dependency.
* **Plugin Support**: Jenkins can be extended via [3rd party plugins](https://wiki.jenkins-ci.org/display/JENKINS/Plugins). You can write plugins to make Jenkins support tools/processes that your team uses.

## Setting up Selenium with Jenkins CI for parallel execution

### Install Jenkins

* Download from <https://jenkins-ci.org/>
* Installation: **java -jar jenkins.war**
* Launch Jenkins: visit <http://localhost:8080/> to get to the dashboard.
* Any configuration that you do with this Jenkins will be stored in ~/.jenkins

### Install and configure Jenkins Selenium Plugin

* Click Manage Jenkins on the Jenkins Dashboard.
* Click on Manager Plugins from the Manage Jenkins option.
* Click on the Available tab.
* Locate and select Selenium Plugin from the list of available plugins.
* Click on the Download now and install after restart button.
* A new screen will be displayed Installing Plugins/Upgrades.
* After the plugin is downloaded, restart Jenkins. Make sure that no jobs are running while you restart.
* After Jenkins restarts, a Selenium Grid link will appear on the left side navigation pane.
* Click on Selenium Grid, the Registered Remote Controls page will be displayed.
* Selenium Grid is now available on http://localhost:4444/wd/hub for tests